All Classes

Attribute
CDF
CDFConstants
CDFData
CDFDelegate
CDFException
CDFNativeLibrary
CDFObject
CDFTools
CDFTT2000
CDFUtils
Entry
Epoch
Epoch16
EpochNative
Variable
This class contains the methods that are associated with either global or variable attributes.

Version:
1.0, 2.0 03/18/05 Selection of current CDF and attribute are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called., 3.3 01/31/11 Reset the vector size to match the maximum entry in stead of its expanded capacity in getEntries method.

See Also:
CDF, CDFException, Entry, Variable
Attribute

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING,
ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING,
ALPHAVMSq_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ATTR_,
ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_,
ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_,
ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_,
BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS,
BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR,
BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_C_NAME,
BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE,
BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL,
BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM,
BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORORITY, BAD_MALLOC,
BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_VARS,
BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM,
BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME,
BAD_VAR_NUM, BAD_zMODE, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE,
CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,
CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN,
CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR,
CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_,
CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR,
CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS,
CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1,
CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_MAJORORITY_,
CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC,
CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_,
CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR,
CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4,
CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_,
CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR,
CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN,
CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR,
CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR,
CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_,
COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF,
CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_,
CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_,
DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING,
DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE,
DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE,
DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE,
DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE,
DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE,
DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE,
DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_,
DID_NOT_COMPRESS, DUMMY_TT2000_VALUE, EMPTY_COMPRESSED_CDF,
END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND,
EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN,
EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN,
EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN,
EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX,
FORCED_PARAMETER, gENTRY__, gENTRY_DATA__, gENTRY_DATASPEC__,
gENTRY_DATATYPE__, gENTRY_EXISTENCE__, gENTRY_NUMELEMS__, GET__,
GETCDFCHECKSUM__, GETCDFFILEBACKWARD__, GETCDFVALIDATE__,
GETLEAPSECONDSENVVAR__, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING,
HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,
IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING,
IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE,
ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF,
ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT__, LIB_INCREMENT__,
LIB_RELEASE__, LIB_subINCREMENT__, LIB_VERSION__, MAC_DECODING,
MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT,
NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on,
NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING,
NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION,
NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS,
NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS,
NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY,
NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF,
NOT_A_CDF OR NOT_SUPPORTED, NOVARY, NULL__, OPEN__,
OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS,
PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED,
PREV_SPARSERECORDS, PUT__, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE,
READONLYoff, READONLYon, rENTRY__, rENTRY_DATA__, rENTRY_DATASPEC__,
rENTRY_DATATYPE__, rENTRY_EXISTENCE__, rENTRY_NAME__, rENTRY_NUMELEMS__,

http://localhost:8080/cdfdocs/gsfc/nssdc/cdf/Attribute.html (3 of 13) [7/12/2011 7:00:53 PM]
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static <code>Attribute</code> create(<code>CDF</code> myCDF, <code>java.lang.String</code> name, <code>long</code> scope)</td>
<td>Creates a new attribute in the given CDF.</td>
</tr>
<tr>
<td><code>void</code> delete()</td>
<td>Deletes this attribute.</td>
</tr>
<tr>
<td><code>void</code> deleteEntry(<code>long</code> entryID)</td>
<td>Deletes an attribute entry for the given entry number.</td>
</tr>
<tr>
<td><code>void</code> deleteEntry(<code>Variable</code> var)</td>
<td>Deletes the attribute entry for the given variable.</td>
</tr>
<tr>
<td><code>java.util.Vector</code> getEntries()</td>
<td>Gets all the entries defined for this attribute.</td>
</tr>
<tr>
<td><code>Entry</code> getEntry(<code>long</code> entryID)</td>
<td>Gets the attribute entry for the given entry number.</td>
</tr>
<tr>
<td><code>Entry</code> getEntry(<code>Variable</code> var)</td>
<td>Gets the attribute entry for the given variable.</td>
</tr>
<tr>
<td><code>long</code> getEntryID(<code>Entry</code> entry)</td>
<td>Gets the entry id for the given entry.</td>
</tr>
<tr>
<td><code>long</code> getID()</td>
<td>Gets the attribute ID of this attribute.</td>
</tr>
<tr>
<td><code>long</code> getMaxEntryNumber()</td>
<td>Gets the largest Entry number for this attribute.</td>
</tr>
<tr>
<td><code>CDF</code> getMyCDF()</td>
<td>Gets the CDF object to which this attribute belongs.</td>
</tr>
<tr>
<td><code>java.lang.String</code> getName()</td>
<td>Gets the name of this attribute.</td>
</tr>
<tr>
<td><code>long</code> getNumEntries()</td>
<td>Gets the number of entries in this attribute.</td>
</tr>
<tr>
<td><code>long</code> getScope()</td>
<td>Gets the scope of this attribute.</td>
</tr>
<tr>
<td><code>void</code> rename(<code>java.lang.String</code> newName)</td>
<td>Renames the current attribute.</td>
</tr>
</tbody>
</table>
java.lang.String **toString** ()

Gets the name of this attribute.

### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

## Method Detail

### create

**public static Attribute create(CDF myCDF,**

```java
    java.lang.String name,
    long scope)
```

**throws CDFException**

Creates a new attribute in the given CDF. Attributes and attribute entries are used to describe information about a CDF file and the variables in the file. Any number of attributes may be stored in a CDF file.

The following example creates a global attribute called 'Project' and a variable attribute called 'VALIDMIN':

```java
    Attribute project, validMin;

    project  = Attribute.create(cdf, "Project", GLOBAL_SCOPE);
    validMin = Attribute.create(cdf, "VALIDMIN", VARIABLE_SCOPE);
```

**Parameters:**

- **myCDF** - the CDF object to which this attribute belongs

- **name** - the name of the attribute to be created

- **scope** - the attribute's scope - it should be either GLOBAL_SCOPE or VARIABLE_SCOPE
**Throws:**

`CDFException` - if a problem occurred in creating an attribute

---

**delete**

```
public void delete()
    throws CDFException
```

Deletes this attribute.

**Note:** When an attribute is deleted all the entries for attribute are deleted as well. Also, all attributes that follow the deleted attribute will be renumbered immediately (their IDs will be decremented by one). This can cause confusion when using a loop to delete attributes. The following is incorrect and will result in every other attribute being deleted:

```
Vector attrs = cdf.getAttributes();
int n = attrs.size();
for (int i = 0 i < n; i++)
    ((Attribute)attrs.getElementAt(i)).delete();
```

Two possible workarounds are:

```
Vector attrs = cdf.getAttributes();
int n = attrs.size();
for (int i = n-1; i >= 0; i--)
    ((Attribute)attrs.getElementAt(i)).delete();
```

and

```
Vector attrs = cdf.getAttributes();
int n = attrs.size();
for (int i = 0 i < n; i++)
    ((Attribute)attrs.getElementAt(0)).delete();
```

**Specified by:**
**delete** in interface **CDFObject**

**Throws:**

**CDFException** - if there is a problem deleting the attribute

---

**getEntry**

```java
public Entry getEntry(long entryID)
throws CDFException
```

Gets the attribute entry for the given entry number.

The following example retrieves the first entry of the global attribute 'project'. Please note that a global attribute can have multiple entries (whereas, a variable attribute has only one entry for a particular attribute), and attribute id starts at 0, not 1.

```java
Entry tEntry = project.getEntry(0L)
```

**Parameters:**

- **entryID** - the entry number from which an attribute entry is retrieved

**Throws:**

**CDFException** - if an error occurred getting an entry (i.e. invalid entryID, no attribute entry for entryID)

---

**getEntry**

```java
public Entry getEntry(Variable var)
throws CDFException
```

Gets the attribute entry for the given variable.

The following example retrieves the 'longitude' variable entry associate with the attribute 'validMin':

```java
vEntry = validMin.getEntry(longitude);
```
Parameters:
  var - the variable from which an attribute entry is retrieved

Throws:
  CDFException - if an error occurred getting a variable attribute entry (e.g. non-existent variable, no attribute entry for this variable, etc.)

```
deleteEntry

public void deleteEntry(long entryID)
    throws CDFException

Deletes an attribute entry for the given entry number.

The following example deletes the first and second entries of the global attribute 'Project':

    project.deleteEntry(0L);
    project.deleteEntry(1L);

The following example deletes the 'longitude' variable entry associated with the attribute 'validMin':

    validMin.deleteEntry(longitude.getID());
```

Parameters:
  entryID - the ID of the entry to be deleted

Throws:
  CDFException - if there was a problem deleting the entry

```
deleteEntry

public void deleteEntry(Variable var)

```
Deletes the attribute entry for the given variable.

The following example deletes the 'longitude' variable entry associated with the attribute 'validMin':

```java
validMin.deleteEntry(longitude);
```

**Parameters:**
- `var` - the variable from which the attribute entry is deleted

**Throws:**
- `CDFException` - if there was a problem deleting the entry

---

### getEntries

`public java.util.Vector getEntries()`

Gets all the entries defined for this attribute. A global attribute can have multiple entries. Whereas, a variable attribute has only one entry for a particular attribute.

**Returns:**
- all the entries (one or more) defined for a global attribute or all variable entries for this attribute. The number of returned entries vector for a variable attribute may be less than the number of variables, as not all variables have an entry for the attribute.

---

### getEntryID

`public long getEntryID(Entry entry)`

Gets the entry id for the given entry.

**Parameters:**
- `entry` - the entry from which an entry id is retrieved
**Returns:**
the entry id for the given entry

---

**rename**

```java
public void rename(java.lang.String newName)
    throws CDFException
```

 Renames the current attribute.

**Specified by:**
rename in interface CDFObject

**Parameters:**
newName - the new attribute name

**Throws:**
CDFException - if there was a problem renaming the attribute

---

**getNumEntries**

```java
public long getNumEntries()
```

 Gets the number of entries in this attribute.

**Returns:**
the number of entries in this attribute

---

**getMaxEntryNumber**

```java
public long getMaxEntryNumber()
```

 Gets the largest Entry number for this attribute.

**Returns:**
**getID**

public long **getID()**

Gets the attribute ID of this attribute.

**Returns:**
the attribute id of this attribute

**getMyCDF**

public **CDF** **getMyCDF()**

Gets the CDF object to which this attribute belongs.

**Returns:**
the CDF object to which this attribute belongs

**getName**

public **java.lang.String** **getName()**

Gets the name of this attribute.

**Specified by:**
getName in interface CDFObject

**Returns:**
the name of this attribute

**toString**
public java.lang.String toString()

    Gets the name of this attribute.

    **Overrides:**
    toString in class java.lang.Object

    **Returns:**
    the name of this attribute

---

**getScope**

public long getScope()

    Gets the scope of this attribute.

    **Returns:**
    If the attribute is a global attribute, GLOBAL_SCOPE is returned. If the attribute is a variable attribute, VARIABLE_SCOPE is returned.
Packages

- gsfc.nssdc.cdf
- gsfc.nssdc.cdf.util
Hierarchy For All Packages

Package Hierarchies:

- gsfc.nssdc.cdf
- gsfc.nssdc.cdf.util

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.CDF (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDOBject)
  - gsfc.nssdc.cdf.CDFData (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDOBject)
  - gsfc.nssdc.cdf.CDFNativeLibrary (implements gsfc.nssdc.cdf.CDFDelegate)
  - gsfc.nssdc.cdf.CDFTools (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.CDFTT2000 (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.CDFUtils (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.Epoch (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.Epoch16 (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.EpochNative
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - gsfc.nssdc.cdf.CDFException (implements gsfc.nssdc.cdf.CDFConstants)

Interface Hierarchy
Class Hierarchy

- gsfc.nssdc.cdf.CDFConstants
- gsfc.nssdc.cdf.CDFDelegate
- gsfc.nssdc.cdf.CDFObject
Deprecated API

Contents

- Deprecated Methods

### Deprecated Methods

**gsfc.nssdc.cdf.CDF.create(String, int)**

Use setFileBackward(long) method to set the file backward flag and create(String) to create file instead.
A

AHUFF_COMPRESSION - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALL_VALUES - Static variable in class gsfc.nssdc.cdf.CDFTools

allocateBlock(long, long) - Method in class gsfc.nssdc.cdf.Variable
Allocates a range of records for this variable.
allocateRecords(long) - Method in class gsfc.nssdc.cdf.Variable
Allocates a number of records, starting from record number 0.

ALPHAOSF1_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAOSF1_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSd_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSd_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSg_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSg_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSi_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ALPHAVMSi_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ATTR_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ATTR_EXISTENCE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ATTR_EXISTS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ATTR_MAXgENTRY_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
**Index**

**ATTR_MAXrENTRY** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_MAXzENTRY** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NAME** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NAME_TRUNC** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NUMBER** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NUMgENTRIES** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NUMrENTRIES** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_NUMzENTRIES** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**ATTR_SCOPE** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**Attribute** - Class in gsfc.nssdc.cdf

This class contains the methods that are associated with either global or variable attributes.

---

**B**

**BACKWARD** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BACKWARDFILEoff** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BACKWARDFILEon** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BAD_ALLOCATE_RECS** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BAD_ARGUMENT** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BAD_ATTR_NAME** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BAD_ATTR_NUM** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**BAD_BLOCKING_FACTOR** - Static variable in interface gsfc.nssdc.cdf. CDFConstants
BAD_CACHE_SIZE - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_CDF_EXTENSION - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_CDF_ID - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_CDF_NAME - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_CDFSTATUS - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_CHECKSUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_COMPRESSION_PARM - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DATA_TYPE - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DIM_COUNT - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DIM_INDEX - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DIM_INTERVAL - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_DIM_SIZE - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_ENTRY_NUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_FNC_OR_ITEM - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_FORMAT - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_INITIAL_RECS - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_MAJORIT - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_MALLOC - Static variable in interface gsfc.nssdc.cdf.CDFConstants
BAD_NEGtoPOSfp0_MODE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_NUM_DIMS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_NUM_ELEMS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_NUM_VARS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_READONLY_MODE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_REC_COUNT - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_REC_INTERVAL - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_REC_NUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_SCOPE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_SCRATCH_DIR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_SPARSEARRAYS_PARM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_VAR_NAME - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_VAR_NUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

BAD_zMODE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

breakdown(long) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000
    Breaks a TT2000 epoch value down into its full component parts.

breakdown(double) - Static method in class gsfc.nssdc.cdf.util.Epoch
    Breaks an EPOCH value down into its component parts.

breakdown(Object) - Static method in class gsfc.nssdc.cdf.util.Epoch16
    Breaks an EPOCH16 value down into its component parts.

breakdown(double) - Static method in class gsfc.nssdc.cdf.util.EpochNative
    Mirrors EPOCHbreakdown from the CDF library.
cannot_allocate_records - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cannot_change - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cannot_compress - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cannot_copy - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cannot_sparsearrays - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cannot_sparerecords - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf - Class in gsfc.nssdc.cdf

The CDF class is the main class used to interact with a CDF file.
cdf_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_access_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_attr_name_len - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_attr_name_len256 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_byte - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_cachesize_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_char - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_checksum_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_close_error - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_compression_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_copyright_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_copyright_len - Static variable in interface gsfc.nssdc.cdf.CDFConstants

cdf_create_error - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_DECODING - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_DELETE_ERROR - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_DOUBLE - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_ENCODING - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_EPOCH - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_EPOCH16 - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_EXISTS - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_FLOAT - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_FORMAT - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INCREMENT - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INFO - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INT1 - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INT2 - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INT4 - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INT8 - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_INTERNAL_ERROR - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_MAJORORITY - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_MAX_DIMS - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_MAX_PARMS - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_MIN_DIMS - Static variable in interface gsfc.nssdc.cdf. CDFConstants

CDF_NAME - Static variable in interface gsfc.nssdc.cdf. CDFConstants
CDF_NAME_TRUNC - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NEGtoPOSfp0_MODE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NUMATTRS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NUMgATTRS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NUMrVARS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NUMvATTRS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_NUMzVARS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_OK - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_OPEN_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_PATHNAME_LEN - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_READONLY_MODE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_REAL4 - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_REAL8 - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_RELEASE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_SAVE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_SCRATCHDIR_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_STATUS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_STATUSTEXT_LEN - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_TIME_TT2000 - Static variable in interface gsfc.nssdc.cdf.CDFConstants
CDF_UCHAR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_UINT1 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_UINT2 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_UINT4 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_VAR_NAME_LEN - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_VAR_NAME_LEN256 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_VERSION - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_WARN - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_WRITE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDF_zMODE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CDFConstants - Interface in gsfc.nssdc.cdf

This class defines the constants used by the CDF library and CDF Java APIs, and it mimics the cdf.h include file from the cdf distribution.

CDFData - Class in gsfc.nssdc.cdf

This class acts as the glue between the Java code and the Java Native Interface (JNI) code.

CDFDelegate - Interface in gsfc.nssdc.cdf

This class defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.

CDFException - Exception in gsfc.nssdc.cdf

This class defines the informational, warning, and error messages that can arise from CDF operations.

CDFException(String) - Constructor for exception gsfc.nssdc.cdf.CDFException

Takes a text message from the calling program and throws a CDFException.

CDFException(long) - Constructor for exception gsfc.nssdc.cdf.CDFException

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.

CDFException(long, String) - Constructor for exception gsfc.nssdc.cdf.CDFException

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.

cdfFileExists(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Checks the existence of the given CDF file name.
CDFgetLastErrorinLeapSecondsTable() - Static method in class gsfc.nssdc.cdf.util.CDFTT2000
This method returns the last date that a leap second was added in the leap second table used in the class.

CDFgetLeapSecondsTable() - Static method in class gsfc.nssdc.cdf.util.CDFTT2000
This method returns the leap seconds table.

CDFgetLeapSecondsTableStatus() - Static method in class gsfc.nssdc.cdf.util.CDFTT2000
This method returns the status code reflecting whether the leap seconds are from a external file, defined by an environment variable, or the leap seconds are based on the hard-coded table in the class.

CDFgetRowsinLeapSecondsTable() - Static method in class gsfc.nssdc.cdf.util.CDFTT2000
This method returns the number of entries in the leap seconds table.

cdflib(CDF, CDFObject, Vector) - Method in interface gsfc.nssdc.cdf.CDFDelegate
Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.

cdflib(CDF, CDFObject, Vector) - Method in class gsfc.nssdc.cdf.CDFNativeLibrary
Calls the Java Native Interface (JNI) program, cdfNativeLibrary.c.

CDFNativeLibrary - Class in gsfc.nssdc.cdf
This class implements the method that act as the gateway between the CDF Java APIs and the CDF library.

CDFNativeLibrary() - Constructor for class gsfc.nssdc.cdf.CDFNativeLibrary

CDFObject - Interface in gsfc.nssdc.cdf
CDFObject provides the base interface for all CDF objects.

CDFTools - Class in gsfc.nssdc.cdf
CDFTools.java Created: Tue Nov 24 16:14:50 1998

CDFTools() - Constructor for class gsfc.nssdc.cdf.CDFTools

CDFTT2000 - Class in gsfc.nssdc.cdf.util

CDFTT2000() - Constructor for class gsfc.nssdc.cdf.util.CDFTT2000

CDFUtils - Class in gsfc.nssdc.cdf.util
This class contains the handy utility routines (methods) called by the core CDF Java APIs.

CDFUtils() - Constructor for class gsfc.nssdc.cdf.util.CDFUtils

CDFwithSTATS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

checkPadValueExistence() - Method in class gsfc.nssdc.cdf.Variable
Checks if the pad value has been defined for this variable.

CHECKSUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants
**CHECKSUM_ERROR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**CHECKSUM_NOT_ALLOWED** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

```java
public void close()
```
Closes this CDF file.

**CLOSE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**COLUMN_MAJOR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**COMPRESS_CACHESIZE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**COMPRESSION_ERROR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

```java
public static void compute(long, long, long, long, long, long, long, long, long)
```
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.

```java
public static void compute(long, long, long, long, long, long, long, long, long, long, Object)
```
Computes an EPOCH16 value based on its component parts.

```java
public static void compute(long, long, long, long, long, long, long, long, long, long, Object)
```
Computes an EPOCH value based on its component parts.

```java
public static void compute(long, long, long, long, long, long, long)
```
Mirrors computeEPOCH from the CDF library.

```java
public void concatenateDataRecords(Variable)
```
Concatenates this variable's data records to the destination variable.

**CONFIRM** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

```java
public void confirmCacheSize()
```
Gets the number of 512-byte cache buffers defined for this variable.

```java
public void confirmCDFCacheSize()
```
Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.

```java
public void confirmCompressCacheSize()
```
Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).

```java
public void confirmDecoding()
```
Gets the CDF decoding method defined for this CDF.

```java
public void confirmNegtoPosfp0()
```
Gets the -0.0 to 0.0 translation flag set for this CDF.

```java
public void confirmPadValue()
```
- Method in class gsfc.nssdc.cdf.Variable
Checks the existence of an explicitly specified pad value for the current z variable.
confirmReadOnlyMode() - Method in class gsfc.nssdc.cdf.CDF
   Gets the value of the read-only mode flag set for this CDF file.
confirmReservePercent() - Method in class gsfc.nssdc.cdf.Variable
   Gets the reserve percentage set for this variable.
confirmStageCacheSize() - Method in class gsfc.nssdc.cdf.CDF
   Gets the number of 512-byte cache buffers defined for the staging scratch file.
confirmzMode() - Method in class gsfc.nssdc.cdf.CDF
   Gets the zMode set for this CDF.
copy(String) - Method in class gsfc.nssdc.cdf.Variable
   Copies this variable to a new variable.
copy(CDF, String) - Method in class gsfc.nssdc.cdf.Variable
   Copies this variable into a new variable and puts it into the designated CDF file.
copyDataRecords(Variable) - Method in class gsfc.nssdc.cdf.Variable
   Copies this variable's data to the destination variable.
CORRUPTED_V2_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CORRUPTED_V3_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

create(CDF, String, long) - Static method in class gsfc.nssdc.cdf.Attribute
   Creates a new attribute in the given CDF.
create(String) - Static method in class gsfc.nssdc.cdf.CDF
   Creates a CDF file in the current directory.
create(String, int) - Static method in class gsfc.nssdc.cdf.CDF
   Deprecated. Use setFileBackward(long) method to set the file backward flag and create(String)
   to create file instead.
create(Attribute, long, long, Object) - Static method in class gsfc.nssdc.cdf.Entry
   Creates a new global or variable attribute entry.
create(CDF, String, long, long, long[], long, long[]) - Static method in class
   gsfc.nssdc.cdf.Variable
   Creates a variable.
CREATE__ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CURgENTRY_EXISTENCE__ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CURrENTRY_EXISTENCE__ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

CURzENTRY_EXISTENCE__ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
**D**

**DATATYPE_MISMATCH** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DATATYPE_SIZE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DECOMPRESSION_ERROR** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DECSTATION_DECODING** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DECSTATION_ENCODING** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_BYTE_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_CHAR_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_DOUBLE_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_EPOCH_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_FLOAT_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_INT1_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_INT2_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_INT4_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_INT8_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_REAL4_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_REAL8_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_TT2000_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_UCHAR_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)

**DEFAULT_UINT1_PADVALUE** - Static variable in interface gsfc.nssdc.cdf. [CDFConstants](#)
DEFAULT_UINT2_PADVALUE - Static variable in interface gsfc.nssdc.cdf. CDFConstants

DEFAULT_UINT4_PADVALUE - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**delete()** - Method in class gsfc.nssdc.cdf. Attribute
   Deletes this attribute.

**delete()** - Method in class gsfc.nssdc.cdf. CDF
   Deletes this CDF file.

**delete()** - Method in class gsfc.nssdc.cdf. CDFData
   See the description of the getName() method in this class.

**delete()** - Method in interface gsfc.nssdc.cdf. CDFObject
   Deletes the current object.

**delete()** - Method in class gsfc.nssdc.cdf. Entry
   Deletes this entry.

**delete()** - Method in class gsfc.nssdc.cdf. Variable
   Deletes this variable.

**DELETE** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**deleteEntry(long)** - Method in class gsfc.nssdc.cdf. Attribute
   Deletes an attribute entry for the given entry number.

**deleteEntry(Variable)** - Method in class gsfc.nssdc.cdf. Attribute
   Deletes the attribute entry for the given variable.

**deleteRecords(long, long)** - Method in class gsfc.nssdc.cdf. Variable
   Deletes a range of records from this variable.

**DID_NOT_COMPRESS** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**DUMMY_TT2000_VALUE** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**dump()** - Method in class gsfc.nssdc.cdf. CDFData
   Dump data information and values, one row at a time, to the stdErr.

**dumpData()** - Method in class gsfc.nssdc.cdf. CDFData
   Dumps variable data, one row at a time per record.

**duplicate(String)** - Method in class gsfc.nssdc.cdf. Variable
   Duplicates this variable to a new variable.

**duplicate(CDF, String)** - Method in class gsfc.nssdc.cdf. Variable
   Duplicates this variable and put it into the designated CDF file.
EMPTY_COMPRESSED_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**encode(long)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Converts an epoch value in TT2000 form into a readable date/time string in ISO 8601 format.

**encode(long, int)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

**encode(double)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time string.

**encode(Object)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

Converts an EPOCH16 value into a readable date/time string.

**decode(double)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors encodeEPOCH from the CDF library.

**decode1(double)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time string.

**decode1(Object)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

Converts an EPOCH16 value into a readable date/time string.

**decode1(double)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors encodeEPOCH1 from the CDF library.

**decode2(double)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time string.

**decode2(Object)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

Converts an EPOCH16 value into a readable date/time string.

**decode2(double)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors encodeEPOCH2 from the CDF library.

**decode3(double)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time string.

**decode3(Object)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

Converts an EPOCH16 value into a readable date/time string.

**decode3(double)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors encodeEPOCH3 from the CDF library.

**decode4(double)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time, ISO8601 string.

**decode4(Object)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

Converts an EPOCH16 value into a readable date/time, ISO8601 string.

**decode4(double)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors encodeEPOCH4 from the CDF library.

**decodex(double, String)** - Static method in class gsfc.nssdc.cdf.util.Epoch

Converts an EPOCH value into a readable date/time string using the specified format.
**Index**

*encodex*(Object, String) - Static method in class gsfc.nssdc.cdf.util. **Epoch16**

Converts an EPOCH16 value into a readable date/time string using the specified format.

*encodex*(double, String) - Static method in class gsfc.nssdc.cdf.util. **EpochNative**

Mirrors encodeEPOCHx from the CDF library.

**END_OF_VAR** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**

**Entry** - Class in gsfc.nssdc.cdf

This class describes a CDF global or variable attribute entry.

**Epoch** - Class in gsfc.nssdc.cdf.util

**Example:** // Get the milliseconds to Aug 5, 1990 at 5:00 double ep = Epoch.compute(1990, 8, 5, 5, 0, 0, 0); //Get the year, month, day, hour, minutes, seconds, milliseconds for ep long times[] = Epoch.breakdown(ep); for (int i=0;i<times.length;i++) System.out.print(times[i]+" ");
System.out.println(); // Printout the epoch in various formats
System.out.println(Epoch.encode(ep)); System.out.println(Epoch.encode1(ep));
System.out.println(Epoch.encode2(ep)); System.out.println(Epoch.encode3(ep));
System.out.println(Epoch.encode4(ep)); // Print out the date using format String format = ", at ";
System.out.println(Epoch.encodex(ep,format));

**Epoch()** - Constructor for class gsfc.nssdc.cdf.util. **Epoch**

**Epoch16** - Class in gsfc.nssdc.cdf.util

**Example:** // Get the time, down to picoseconds, for Aug 5, 1990 at 5:0:0.0.0.0 double[] epoch16 = new double[2]; double ep = Epoch16.compute(1990, 8, 5, 5, 0, 0, 0, 0, 0, 0, epoch16); //Get the year, month, day, hour, minutes, seconds, milliseconds, // microseconds, nanoseconds and picoseconds for epoch16 long times[] = Epoch16.breakdown(epoch16); for (int i=0;i<times.length;i++) System.out.print(times[i]+" "); System.out.println(); // Printout the epoch in various formats System.out.println(Epoch16.encode(epoch16));
System.out.println(Epoch16.encode1(epoch16)); System.out.println(Epoch16.encode2(epoch16));
System.out.println(Epoch16.encode3(epoch16)); System.out.println(Epoch16.encode4(epoch16)); // Print out the date using format String format = ", at ";
System.out.println(Epoch16.encodex(epoch16,format));

**Epoch16()** - Constructor for class gsfc.nssdc.cdf.util. **Epoch16**

**EPOCH1_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**

**EPOCH1_STRING_LEN_EXTEND** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**

**EPOCH2_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**

**EPOCH2_STRING_LEN_EXTEND** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**

**EPOCH3_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf. **CDFConstants**
EPOCH3_STRING_LEN_EXTEND - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EPOCH4_STRING_LEN - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EPOCH4_STRING_LEN_EXTEND - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EPOCH_STRING_LEN - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EPOCH_STRING_LEN_EXTEND - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EpochNative - Class in gsfc.nssdc.cdf.util

The Epoch class is a Java wrapper to the CDF epoch handling routines.

EpochNative() - Constructor for class gsfc.nssdc.cdf.util.EpochNative

EPOCHx_FORMAT_MAX - Static variable in interface gsfc.nssdc.cdf.CDFConstants

EPOCHx_STRING_MAX - Static variable in interface gsfc.nssdc.cdf.CDFConstants

finalize() - Method in class gsfc.nssdc.cdf.CDF

Do the necessary cleanup when garbage collector reaps it.

FORCED_PARAMETER - Static variable in interface gsfc.nssdc.cdf.CDFConstants

fromGregorianTime(GregorianCalendar) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

This method converts the date/time in a GregorianCalendar class object to TT2000 time.

fromUTCEPOCH(double) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Convert an epoch value in CDF_EPOCH to TT2000.

fromUTCEPOCH16(double[]) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Convert an epoch data in CDF_EPOCH16 to TT2000.

fromUTCparts(double, double, double) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.

fromUTCparts(double, double, double, double) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.

fromUTCparts(double, double, double, double, double) - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. `fromUTCparts(double, double, double, double, double, double, double)` - Static method in class `gsfc.nssdc.cdf.util.CDFTT2000`

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. `fromUTCparts(double, double, double, double, double, double, double, double)` - Static method in class `gsfc.nssdc.cdf.util.CDFTT2000`

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. `fromUTCparts(double, double, double, double, double, double, double, double, double)` - Static method in class `gsfc.nssdc.cdf.util.CDFTT2000`

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. `fromUTCparts(double, double, double, double, double, double, double, double, double)` - Static method in class `gsfc.nssdc.cdf.util.CDFTT2000`

This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000.

**G**

`gENTRY_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`gENTRY_DATA_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`gENTRY_DATASPEC_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`gENTRY_DATATYPE_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`gENTRY_EXISTENCE_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`gENTRY_NUMELEMS_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`GET_` - Static variable in interface `gsfc.nssdc.cdf.CDFConstants`

`getAllocatedFrom(long)` - Method in class `gsfc.nssdc.cdf.Variable`

Inquires the next allocated record at or after a given record for this variable.

`getAllocatedTo(long)` - Method in class `gsfc.nssdc.cdf.Variable`

Inquires the last allocated record (before the next unallocated record) at or after a given record for this variable.
getAttribute(long) - Method in class gsfc.nssdc.cdf.CDF
  Gets the attribute for the given attribute number.

getAttribute(String) - Method in class gsfc.nssdc.cdf.CDF
  Gets the attribute for the given attribute name.

getAttributeID(String) - Method in class gsfc.nssdc.cdf.CDF
  Gets the id of the given attribute.

getAttributes() - Method in class gsfc.nssdc.cdf.CDF
  Gets all the global and variable attributes defined for this CDF.

getAttributes() - Method in class gsfc.nssdc.cdf.Variable
  Returns the variable attributes that are associated with this variable.

getBlockingFactor() - Method in class gsfc.nssdc.cdf.Variable
  Gets the blocking factor for this variable.

GETCDFCHECKSUM_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

GETCDFFILEBACKWARD_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

GETCDFVALIDATE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

getChecksum() - Method in class gsfc.nssdc.cdf.CDF
  Gets the checksum method, if any, applied to the CDF.

getChecksumEnvVar() - Static method in class gsfc.nssdc.cdf.CDF
  Gets the value of the CDF_CHECKSUM environment variable.

getCompression() - Method in class gsfc.nssdc.cdf.CDF
  Gets the string representation of the compression type and parameters defined for this CDF.

getCompression() - Method in class gsfc.nssdc.cdf.Variable
  Gets the string representation of the compression type and parameters set for this variable.

getCompressionParms() - Method in class gsfc.nssdc.cdf.CDF
  Gets the compression parameters set for this CDF.

getCompressionParms() - Method in class gsfc.nssdc.cdf.Variable
  Sets the compression parameters of this variable.

getCompressionPct() - Method in class gsfc.nssdc.cdf.CDF
  Gets the compression percentage set for this CDF.

getCompressionPct() - Method in class gsfc.nssdc.cdf.Variable
  Gets the compression percentage rate of this variable.

getCompressionType() - Method in class gsfc.nssdc.cdf.CDF
  Gets the compression type set for this CDF.

getCompressionType() - Method in class gsfc.nssdc.cdf.Variable
  Gets the compression type of this variable.

getCopyright() - Method in class gsfc.nssdc.cdf.CDF
  Gets the CDF copyright statement for this CDF.

getCurrentStatus() - Method in exception gsfc.nssdc.cdf.CDFException
Gets the status code that caused CDFException.

**getData()** - Method in class gsfc.nssdc.cdf.CDFData

Returns an object that is properly dimensioned.

**getData()** - Method in class gsfc.nssdc.cdf.Entry

Gets the data for this entry.

**getDataType()** - Method in class gsfc.nssdc.cdf.Entry

Gets the CDF data type of this entry.

**getDataType()** - Method in class gsfc.nssdc.cdf.Variable

Gets the CDF data type of this variable.

**getDataTypeValue(String)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Gets the long value of the given CDF data type in string.

**getDelegate()** - Method in class gsfc.nssdc.cdf.CDF

This is a placeholder for future expansions/extensions.

**getDimCounts()** - Method in class gsfc.nssdc.cdf.CDFData

Gets the value of the dimension counts that represents the number of elements read or write starting at the location for a hyper get/put function.

**getDimIndices()** - Method in class gsfc.nssdc.cdf.CDFData

Gets the starting dimension index within a record for a hyper get/put function.

**getDimIntervals()** - Method in class gsfc.nssdc.cdf.CDFData

Gets the value of the dimension intervals that represent the number of elements to skip between reads or writes for a hyper get/put function.

**getDimSizes()** - Method in class gsfc.nssdc.cdf.CDFData

Gets the dimension sizes of this variable.

**getDimSizes()** - Method in class gsfc.nssdc.cdf.Variable

Gets the dimensions size of this variable.

**getDimVariances()** - Method in class gsfc.nssdc.cdf.Variable

Gets the dimension variances for this variable.

**getEncoding()** - Method in class gsfc.nssdc.cdf.CDF

Gets the encoding method defined for this CDF.

**getEntries()** - Method in class gsfc.nssdc.cdf.Attribute

Gets all the entries defined for this attribute.

**getEntry(long)** - Method in class gsfc.nssdc.cdf.Attribute

Gets the attribute entry for the given entry number.

**getEntry(Variable)** - Method in class gsfc.nssdc.cdf.Attribute

Gets the attribute entry for the given variable.

**getEntryData(String)** - Method in class gsfc.nssdc.cdf.Variable

Gets the attribute entry data for this variable.

**getEntryID(Entry)** - Method in class gsfc.nssdc.cdf.Attribute

Gets the entry id for the given entry.

**getFileBackward()** - Static method in class gsfc.nssdc.cdf.CDF

Gets the file backward flag.
getFileBackwardEnvVar() - Static method in class gsfc.nssdc.cdf.CDF
Gets the value of the CDF_FILEBACKWARD environment variable.

getFormat() - Method in class gsfc.nssdc.cdf.CDF
Gets the CDF format defined for this CDF.

getGlobalAttributes() - Method in class gsfc.nssdc.cdf.CDF
Gets the global attributes defined for this CDF.

getHyperData(long, long, long, long[], long[], long[]) - Method in class gsfc.nssdc.cdf.Variable
Reads one or more values from the current z variable.

getHyperDataObject(long, long, long, long[], long[], long[]) - Method in class gsfc.nssdc.cdf.Variable
Reads one or more values from the current z variable.

getID() - Method in class gsfc.nssdc.cdf.Attribute
Gets the attribute ID of this attribute.

getID() - Method in class gsfc.nssdc.cdf.CDF
Gets the id of this CDF file.

getID() - Method in class gsfc.nssdc.cdf.Entry
Gets the ID of this entry.

getID() - Method in class gsfc.nssdc.cdf.Variable
Gets the ID of this variable.

GETLEAPSECONDSENVVAR_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

getLeapSecondsTableEnvVar() - Static method in class gsfc.nssdc.cdf.CDF
Gets the the CDF_LEAPSECONDSTABLE (or CDF$LEAPSECONDSTABLE on VMS) environment variable.

getLibraryCopyright() - Static method in class gsfc.nssdc.cdf.CDF
Retrieve library copyright information associated with the CDF library.

getLibraryVersion() - Static method in class gsfc.nssdc.cdf.CDF
Retrieve library version/release/increment/sub_increment information associated with the CDF library.

getLongChecksum(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long value of the given CDF's checksum in string.

getLongCompressionType(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long representation of the given CDF compression type in string.

getLongEncoding(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long value of the given CDF encoding type in string.

getLongFormat(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long value of the given CDF file format in string.

getLongMajority(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long value of the given CDF majority.

getLongSparseRecord(String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
Gets the long value of the given sparse record type in string.

getMajority() - Method in class gsfc.nssdc.cdf.CDF
Gets the variable majority defined for this CDF.

**getMaxAllocatedRecord()** - Method in class gsfc.nssdc.cdf.**Variable**

Gets the maximum allocated record number for this variable.

**getMaxEntryNumber()** - Method in class gsfc.nssdc.cdf.**Attribute**

Gets the largest Entry number for this attribute.

**getMaxWrittenRecord()** - Method in class gsfc.nssdc.cdf.**Variable**

Gets the last written record number, beginning with 0.

**getMyCDF()** - Method in class gsfc.nssdc.cdf.**Attribute**

Gets the CDF object to which this attribute belongs.

**getMyCDF()** - Method in class gsfc.nssdc.cdf.**Variable**

 Gets the CDF object to which this variable belongs.

**getName()** - Method in class gsfc.nssdc.cdf.**Attribute**

Gets the name of this attribute.

**getName()** - Method in class gsfc.nssdc.cdf.**CDF**

 Gets the name of this CDF.

**getName()** - Method in class gsfc.nssdc.cdf.**CDFData**

CDFData implements CDFObject to enable CDFDelegate calls.

**getName()** - Method in interface gsfc.nssdc.cdf.**CDFObject**

Returns the name of the current object.

**getName()** - Method in class gsfc.nssdc.cdf.**Entry**

 Gets the name of this entry.

**getName()** - Method in class gsfc.nssdc.cdf.**Variable**

 Gets the name of this variable.

**getnDims()** - Method in class gsfc.nssdc.cdf.**CDFData**

 Gets the dimensionality of this variable.

**getNumAllocatedRecords()** - Method in class gsfc.nssdc.cdf.**Variable**

 Gets the number of records allocated for this variable.

**getNumAttrs()** - Method in class gsfc.nssdc.cdf.**CDF**

 Gets the total number of global and variable attributes in this CDF.

**getNumDims()** - Method in class gsfc.nssdc.cdf.**Variable**

 Gets the number of dimensions for this variable.

**getNumElements()** - Method in class gsfc.nssdc.cdf.**Entry**

 Gets the number of elements in this entry.

**getNumElements(long, Object)** - Static method in class gsfc.nssdc.cdf.util.**CDFUtils**

 Gets the number of elements contained in the given data object.

**getNumElements()** - Method in class gsfc.nssdc.cdf.**Variable**

 Gets the number of elements for this variable.

**getNumEntries()** - Method in class gsfc.nssdc.cdf.**Attribute**

Gets the number of entries in this attribute.

**getNumGattrs()** - Method in class gsfc.nssdc.cdf.**CDF**

 Gets the number of global attributes in this CDF.
getNumRvars() - Method in class gsfc.nssdc.cdf.CDF

Gets the number of r variables.

getNumVars() - Method in class gsfc.nssdc.cdf.CDF

Gets the number of Z variables defined for this CDF.

getNumVattrs() - Method in class gsfc.nssdc.cdf.CDF

Gets the number of variable attributes in this CDF.

getNumWrittenRecords() - Method in class gsfc.nssdc.cdf.Variable

Gets the number of records physically written (not allocated) for this variable.

getNumZvars() - Method in class gsfc.nssdc.cdf.CDF

Gets the number of z variables in this CDF file.

getOrphanAttributes() - Method in class gsfc.nssdc.cdf.CDF

Gets the variable attributes defined for this CDF that are not associated with any variables.

getPadValue() - Method in class gsfc.nssdc.cdf.Variable

Gets the pad value set for this variable.

getRawData() - Method in class gsfc.nssdc.cdf.CDFData

Returns an object of a 1-dimensional array, which presents a sequence of raw data values retrieved and presented by JNI from a CDF file.

getRecCount() - Method in class gsfc.nssdc.cdf.CDFData

Gets the number of records to read or write for a hyper get/put function.

getRecInterval() - Method in class gsfc.nssdc.cdf.CDFData

Gets the number of records to skip for a hyper get/put function.

getRecord(long, String[]) - Method in class gsfc.nssdc.cdf.CDF

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

getRecord(long, String[], long[]) - Method in class gsfc.nssdc.cdf.CDF

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

getRecord(long, long[]) - Method in class gsfc.nssdc.cdf.CDF

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

getRecord(long, long[], long[]) - Method in class gsfc.nssdc.cdf.CDF

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

getRecord(long) - Method in class gsfc.nssdc.cdf.Variable

Gets a single record from this variable.

getRecordObject(long) - Method in class gsfc.nssdc.cdf.Variable

Gets a single record of data from this variable.

getRecordsObject(long, long) - Method in class gsfc.nssdc.cdf.Variable

Get a number of records of data from this variable.

getRecStart() - Method in class gsfc.nssdc.cdf.CDFData

Gets the record number at which a hyper get/put function starts.
getRecVariance() - Method in class gsfc.nssdc.cdf.Variable
  Gets the value of record variance.

getScalarData() - Method in class gsfc.nssdc.cdf.Variable
  Gets the scalar data from a non-record varying 0-dimensional variable.

getScalarData(long) - Method in class gsfc.nssdc.cdf.Variable
  Get the scalar data from a record varying 0-dimensional variable.

getScalarDataObject() - Method in class gsfc.nssdc.cdf.Variable
  Get the scalar data from a non-record varying 0-dimensional variable.

getScalarDataObject(long) - Method in class gsfc.nssdc.cdf.Variable
  Get the scalar data from this record varying 0-dimensional variable.

gtScope() - Method in class gsfc.nssdc.cdf.Attribute
  Gets the scope of this attribute.

gtSignature(Object) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the java signature of the given object.

gtSingleData(long, long[]) - Method in class gsfc.nssdc.cdf.Variable
  Gets a single data value.

getSingleDataObject(long, long[]) - Method in class gsfc.nssdc.cdf.Variable
  Gets a single data object from this variable.

getSparseRecords() - Method in class gsfc.nssdc.cdf.Variable
  Gets the sparse record type for this variable.

getStatus() - Method in class gsfc.nssdc.cdf.CDF
  Gets the status of the most recent CDF JNI/library function call.

getStatusMsg(long) - Static method in exception gsfc.nssdc.cdf.CDFException
  Get the status text message for the given status code.

getStatusText(long) - Static method in class gsfc.nssdc.cdf.CDF
  Gets the status text of the most recent CDF JNI/library function call.

getStringChecksum(CDF) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the string value of the given CDF's checksum.

getStringChecksum(long) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the string value of the given CDF's checksum.

getStringCompressionType(long) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the string representation of the given CDF compression type.

getStringCompressionType(Variable) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the string representation of the given variable's compression type.

getStringCompressionType(CDF) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Gets the string representation of the given CDF file's compression type.

getStringData(Object) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Returns the string value of the given data.

getStringData(Object, int) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Returns the string value of the given data.

getStringData(Object, String) - Static method in class gsfc.nssdc.cdf.util.CDFUtils
  Returns the string value of the given data.
returns the string of the value of the given data.

**getStringData(Object, String, int)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the CDF data type for the given variable.

**getStringDataType(Variable)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the CDF data type for the given entry.

**getStringDataType(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string representation of the given CDF data type.

**getStringDecoding(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF decoding type.

**getStringDecoding(CDF)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF file's decoding type.

**getStringEncoding(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF encoding type.

**getStringEncoding(CDF)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF's encoding type.

**getStringFormat(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF's file format.

**getStringFormat(CDF)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF's file format.

**getStringMajority(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF majority.

**getStringMajority(CDF)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given CDF file's majority.

**getStringSparseRecord(long)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given sparse record type.

**getStringSparseRecord(Variable)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

gets the string value of the given variable's sparse record type.

**getValidate()** - Static method in class gsfc.nssdc.cdf.CDF

gets the file validation mode.

**getVariable(long)** - Method in class gsfc.nssdc.cdf.CDF

gets the variable object for the given variable number.

**getVariable(String)** - Method in class gsfc.nssdc.cdf.CDF

gets the variable object for the given variable name.

**getVariableAttributes()** - Method in class gsfc.nssdc.cdf.CDF

gets the variable attributes defined for this CDF.

**getVariableID(String)** - Method in class gsfc.nssdc.cdf.CDF

gets the ID of the given variable.

**getVariables()** - Method in class gsfc.nssdc.cdf.CDF

gets the z variables defined for this CDF.
**getVersion()** - Method in class gsfc.nssdc.cdf.CDF

Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).

**GLOBAL_SCOPE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**gsfc.nssdc.cdf** - package gsfc.nssdc.cdf

**gsfc.nssdc.cdf.util** - package gsfc.nssdc.cdf.util

**GZIP_COMPRESSION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**H**

**HOST_DECODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**HOST_ENCODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**HP_DECODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**HP_ENCODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**HUFF_COMPRESSION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**I**

**IBM_PC_OVERFLOW** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**IBMPC_DECODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**IBMPC_ENCODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**IBMRS_DECODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**IBMRS_ENCODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants
ILLEGAL_EPOCH_FIELD - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ILLEGAL_EPOCH_VALUE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ILLEGAL_FOR_SCOPE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ILLEGAL_IN_zMODE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ILLEGAL_ON_V1_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

ILLEGAL_TT2000_VALUE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

IS_A_NETCDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

LIB_COPYRIGHT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

LIB_INCREMENT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

LIB_RELEASE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

LIB_subINCREMENT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

LIB_VERSION_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

MAC_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

MAC_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants
MD5_CHECKSUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

MULTI_FILE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

MULTI_FILE_FORMAT - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NA_FOR_VARIABLE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NAMED_VALUES - Static variable in class gsfc.nssdc.cdf.CDFTools

NEGATIVE_FP_ZERO - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NEGtoPOSfp0off - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NEGtoPOSfp0on - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NETWORK_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NETWORK_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NeXT_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NeXT_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_ATTR_SELECTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_CDF_SELECTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_CHECKSUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_COMPRESSION - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_DELETE_ACCESS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_ENTRY_SELECTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants
NO_MORE_ACCESS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_PADVALUE_SPECIFIED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_REPORTS - Static variable in class gsfc.nssdc.cdf.CDFTools

NO_SPARSEARRAYS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SPARSERECORDS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_STATUS_SELECTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SUCH_ATTR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SUCH_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SUCH_ENTRY - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SUCH_RECORD - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_SUCH_VAR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_VALUES - Static variable in class gsfc.nssdc.cdf.CDFTools

NO_VAR_SELECTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_VARS_IN_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NO_WRITE_ACCESS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NONE_CHECKSUM - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NOT_A_CDF - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NOT_A_CDF_OR_NOT_SUPPORTED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NOVARY - Static variable in interface gsfc.nssdc.cdf.CDFConstants

NRV_VALUES - Static variable in class gsfc.nssdc.cdf.CDFTools
**NULL_** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**open(String)** - Static method in class gsfc.nssdc.cdf. CDF

Open a CDF file for read/write, the default mode for opening a CDF.

**open(String, long)** - Static method in class gsfc.nssdc.cdf. CDF

Open a CDF file.

**OPEN_** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**OPTIMAL_ENCODING_TREES** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**OTHER_CHECKSUM** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**O**

**parse(String)** - Static method in class gsfc.nssdc.cdfutil. CDFTT2000

This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000.

**parse(String)** - Static method in class gsfc.nssdc.cdfutil. Epoch

This function parses an input date/time string and returns an EPOCH value.

**parse(String)** - Static method in class gsfc.nssdc.cdfutil. Epoch16

This function parses an input date/time string and returns an EPOCH16 value.

**parse(String)** - Static method in class gsfc.nssdc.cdfutil. EpochNative

Mirrors parseEPOCH from CDF library.

**parse1(String)** - Static method in class gsfc.nssdc.cdfutil. Epoch

This function parses an input date/time string and returns an EPOCH value.

**parse1(String)** - Static method in class gsfc.nssdc.cdfutil. Epoch16

This function parses an input date/time string and returns an EPOCH16 value.

**parse1(String)** - Static method in class gsfc.nssdc.cdfutil. EpochNative

Mirrors parseEPOCH from CDF library.

**P**

**PAD_SPARSERECORDS** - Static variable in interface gsfc.nssdc.cdf. CDFConstants

**parse(String)** - Static method in class gsfc.nssdc.cdfutil. CDFTT2000
**parse2(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch

This function parses an input date/time string and returns an EPOCH value.

**parse2(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch

This function parses an input date/time string and returns an EPOCH16 value.

**parse2(String)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors parseEPOCH from CDF library.

**parse3(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch

This function parses an input date/time string and returns an EPOCH value.

**parse3(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

This function parses an input date/time string and returns an EPOCH16 value.

**parse3(String)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors parseEPOCH from CDF library.

**parse4(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch

This function parses an input date/time string and returns an EPOCH value.

**parse4(String)** - Static method in class gsfc.nssdc.cdf.util.Epoch16

This function parses an input date/time, ISO8601 string and returns an EPOCH16 value.

**parse4(String)** - Static method in class gsfc.nssdc.cdf.util.EpochNative

Mirrors parseEPOCH from CDF library.

**PPC_DECODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**PPC_ENCODING** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**PRECEEDING_RECORDS_ALLOCATED** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**PREV_SPARSERECORDS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**printData(Object)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data on the screen.

**printData(Object, int, boolean)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data on the screen.

**printData(Object, int)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data on the screen.

**printData(Object, PrintWriter)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.

**printData(Object, PrintWriter, int)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.

**printData(Object, PrintWriter, int, boolean)** - Static method in class gsfc.nssdc.cdf.util.CDFUtils

Prints the value of the given data to the place designated by PrintWriter that can be a file,
System.out, System.err, and etc.

**PUT_** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**putData(long, Object)** - Method in class gsfc.nssdc.cdf.Entry

Put the entry data into the CDF.

**putEntry(String, long, Object)** - Method in class gsfc.nssdc.cdf.Variable

Creates an attribute entry for this variable.

**putEntry(Attribute, long, Object)** - Method in class gsfc.nssdc.cdf.Variable

Creates an attribute entry for this variable.

**putHyperData(long, long, long[], long[], long[], Object)** - Method in class gsfc.nssdc.cdf.Variable

Creates an attribute entry for this variable.

**putHyperData(long, long, long[], long[], long[], Object)** - Method in class gsfc.nssdc.cdf.Variable

Writes one or more values from the current z variable.

**putRecord(long, String[], Vector)** - Method in class gsfc.nssdc.cdf.CDF

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

**putRecord(long, String[], Vector, long[])** - Method in class gsfc.nssdc.cdf.CDF

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

**putRecord(long, long[], Vector)** - Method in class gsfc.nssdc.cdf.CDF

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

**putRecord(long, long[], Vector, long[])** - Method in class gsfc.nssdc.cdf.CDF

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.

**putRecord(long, Object)** - Method in class gsfc.nssdc.cdf.Variable

Adds a single record to a record-varying variable.

**putRecord(Object)** - Method in class gsfc.nssdc.cdf.Variable

Adds a single record to a non-record-varying variable.

**putScalarData(long, Object)** - Method in class gsfc.nssdc.cdf.Variable

Adds a scalar data to this variable (of 0 dimensional).

**putScalarData(Object)** - Method in class gsfc.nssdc.cdf.Variable

Adds a scalar data to this variable (of 0 dimensional).

**putSingleData(long, long[], Object)** - Method in class gsfc.nssdc.cdf.Variable

Adds a single data value to this variable.

**READ_ONLY_DISTRIBUTION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants
**READ ONLY MODE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**READONLYoff** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**READONLYon** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rename(String)** - Method in class gsfc.nssdc.cdf.Attribute

   Renames the current attribute.

**rename(String)** - Method in class gsfc.nssdc.cdf.CDF

   Renames the current CDF.

**rename(String)** - Method in class gsfc.nssdc.cdf.CDFData

   See the description of the getName() method in this class.

**rename(String)** - Method in interface gsfc.nssdc.cdf.CDFObj ect

   Renames the current object.

**rename(String)** - Method in class gsfc.nssdc.cdf.Entry

   This method is here as a placeholder since the Entry class implements the CDFObj ect interface that includes "rename".

**rename(String)** - Method in class gsfc.nssdc.cdf.Variable

   Renames the current variable.

**rENTRY** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_DATA** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_DATASPEC** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_DATATYPE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_EXISTENCE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_NAME** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rENTRY_NUMELEMS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**REPORT_ERRORS** - Static variable in class gsfc.nssdc.cdf.CDFTools

**REPORT_INFORMATION** - Static variable in class gsfc.nssdc.cdf.CDFTools

**REPORT_WARNINGS** - Static variable in class gsfc.nssdc.cdf.CDFTools

**RLE_COMPRESSION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants
Index

**RLE_OF_ZEROS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**ROW_MAJOR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**RV_VALUES** - Static variable in class gsfc.nssdc.cdf.CDFTools

**rVAR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_ALLOCATEBLOCK** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_ALLOCATEDFROM** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_ALLOCATEDTO** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_ALLOCATERECS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_BLOCKINGFACTOR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_CACHESIZE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_COMPRESSION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_DATA** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_DATASPEC** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_DATATYPE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_DIMVARYS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_EXISTENCE** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_HYPERDATA** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_INITIALRECS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_MAXallocREC** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**rVAR_MAXREC** - Static variable in interface gsfc.nssdc.cdf.CDFConstants
Index

rVAR_NAME_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_nINDEXENTRIES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_nINDEXLEVELS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_nINDEXRECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_NUMAllocRECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_NUMBER_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_NUMELEMS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_NUMRECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_PADVALUE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_RECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_RECvARY_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_RESERVEPERCENT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_SEQDATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_SEQPOS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_SPARSEARRAYS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVAR_SPARSERECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_CACHESIZE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_DIMCOUNTS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_DIMINDICES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_DIMINTERVALS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_DIMSIZES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
rVARs_MAXREC_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_NUMDIMS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_RECCOUNT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_RECDATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_RECINTERVAL_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

rVARs_RECNUMBER_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

---

S

save() - Method in class gsfc.nssdc.cdf.CDF
Saves this CDF file without closing.

SAVE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

SCRATCH_CREATE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

SCRATCH_DELETE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

SCRATCH_READ_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

SCRATCH_WRITE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

SELECT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

selectCacheSize(long) - Method in class gsfc.nssdc.cdf.Variable
Sets the number of 512-byte cache buffers to be used.

selectCDFCacheSize(long) - Method in class gsfc.nssdc.cdf.CDF
Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF).

selectCompressCacheSize(long) - Method in class gsfc.nssdc.cdf.CDF
Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF).
Index

**selectDecoding(long)** - Method in class gsfc.nssdc.cdf.CDF

Defines the CDF decoding method to be used for this CDF.

**selectNegtoPosfp0(long)** - Method in class gsfc.nssdc.cdf.CDF

Defines whether to translate -0.0 to 0.0 for reading or writing.

**selectReadOnlyMode(long)** - Method in class gsfc.nssdc.cdf.CDF

Sets the desired read-only mode.

**selectReservePercent(long)** - Method in class gsfc.nssdc.cdf.Variable

Sets the reserve percentage to be used for this variable.

**selectStageCacheSize(long)** - Method in class gsfc.nssdc.cdf.CDF

Sets the number of 512-byte cache buffers to be used for the staging scratch file (for the current CDF).

**setBlockingFactor(long)** - Method in class gsfc.nssdc.cdf.Variable

Sets the blocking factor for this variable.

**setChecksum(long)** - Method in class gsfc.nssdc.cdf.CDF

Specifies the checksum option applied to the CDF.

**setCompression(long, long[])** - Method in class gsfc.nssdc.cdf.CDF

Sets the compression type and parameters for this CDF.

**setDelegate(CDFDelegate)** - Method in class gsfc.nssdc.cdf.CDF

This is a placeholder for future expansions/extensions.

**setDimVariances(long[])** - Method in class gsfc.nssdc.cdf.Variable

Sets the dimension variances for this variable.

**setEncoding(long)** - Method in class gsfc.nssdc.cdf.CDF

Defines the encoding method to be used for this CDF.

**setFileBackward(long)** - Static method in class gsfc.nssdc.cdf.CDF

Sets the file backward flag so that when a new CDF file is created, it will be created in either in the older V2.7 version or the current library version, i.e., V3.*.

**setFormat(long)** - Method in class gsfc.nssdc.cdf.CDF

Specifies the format of this CDF.

**setInfoWarningOff()** - Method in class gsfc.nssdc.cdf.CDF

Sets the informational (status code > 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function off.

**setInfoWarningOn()** - Method in class gsfc.nssdc.cdf.CDF

Sets the informational (status code > 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function on.

**setInitialRecords(long)** - Method in class gsfc.nssdc.cdf.Variable

Sets the number of records to be written initially for this variable.

**setMajority(long)** - Method in class gsfc.nssdc.cdf.CDF

Sets the variable majority for this CDF.

**setPadValue(Object)** - Method in class gsfc.nssdc.cdf.Variable

http://localhost:8080/cdfdocs/index-all.html (36 of 43) [7/12/2011 7:00:58 PM]
Sets the pad value for this variable.

\textbf{setRecVariance(long)} - Method in class \texttt{gsfc.nssdc.cdf.Variable}

Sets the record variance for this variable.

\textbf{setSparseRecords(long)} - Method in class \texttt{gsfc.nssdc.cdf.Variable}

Sets the sparse record type for this variable.

\textbf{setValidate(long)} - Static method in class \texttt{gsfc.nssdc.cdf.CDF}

Sets the file validation mode so that when a CDF file is open, it will be validated accordingly.

\textbf{SGi_DECODING} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{SGi_ENCODING} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{SINGLE_FILE} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{SINGLE_FILE_FORMAT} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{skeletonCDF(String, String, boolean, boolean, boolean, boolean, int, int, int)} - Static method in class \texttt{gsfc.nssdc.cdf.CDFTools}

\texttt{skeletonTable} produces a skeleton table from a CDF.

\textbf{skeletonTable(String, String, boolean, boolean, boolean, boolean, boolean, boolean, boolean, int, String[], int, int, int)} - Static method in class \texttt{gsfc.nssdc.cdf.CDFTools}

\texttt{skeletonTable} produces a skeleton table from a CDF.

\textbf{SOME_ALREADY_ALLOCATED} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{STAGE_CACHESIZE} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{STATUS_TEXT} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{SUN_DECODING} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{SUN_ENCODING} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}

\textbf{toGregorianTime(long)} - Static method in class \texttt{gsfc.nssdc.cdf.util.CDFTT2000}

This method converts the date/time in TT2000 to a GregorianCalendar class object.

\textbf{TOO_MANY_PARMS} - Static variable in interface \texttt{gsfc.nssdc.cdf.CDFConstants}
**TOO_MANY_VARS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**toString()** - Method in class gsfc.nssdc.cdf.Attribute

  Gets the name of this attribute.

**toString()** - Method in class gsfc.nssdc.cdf.CDF

  Gets the name of this CDF.

**toString()** - Method in class gsfc.nssdc.cdf.Variable

  Gets the name of this variable.

**toUTCEPOCH(long)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Convert an epoch in TT2000 value to CDF_EPOCH value.

**toUTCEPOCH16(long, double[])** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Convert an epoch in TT2000 value to CDF_EPOCH16 value.

**toUTCparts(long)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Breaks a TT2000 epoch value down into its full component parts.

**toUTCstring(Long)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

**toUTCstring(long)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

**toUTCstring(Long, int)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

**toUTCstring(long, int)** - Static method in class gsfc.nssdc.cdf.util.CDFTT2000

  Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

**TT2000_0_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**TT2000_1_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**TT2000_2_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**TT2000_3_STRING_LEN** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**TT2000_TIME_ERROR** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**U**

**UNKNOWN_COMPRESSION** - Static variable in interface gsfc.nssdc.cdf.CDFConstants

**UNKNOWN_SPARSENESS** - Static variable in interface gsfc.nssdc.cdf.CDFConstants
UNSUPPORTED_OPERATION - Static variable in interface gsfc.nssdc.cdf.CDFConstants

updateDataSpec(long, long) - Method in class gsfc.nssdc.cdf.Entry
   Update the data specification (data type and number of elements) of the entry.
updateDataSpec(long, long) - Method in class gsfc.nssdc.cdf.Variable
   Update the data specification (data type and number of elements) of the variable.

V

VALIDATE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VALIDATEFILEoff - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VALIDATEFILEon - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_ALREADY_CLOSED - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_CLOSE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_CREATE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_DELETE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_EXISTS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_NAME_TRUNC - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_OPEN_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_READ_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_SAVE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAR_WRITE_ERROR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

Variable - Class in gsfc.nssdc.cdf
   The Variable class defines a CDF variable.
VARIABLE SCOPE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VARY - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAX_DECODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

VAX_ENCODING - Static variable in interface gsfc.nssdc.cdf.CDFConstants

verifyChecksum() - Method in class gsfc.nssdc.cdf.CDF

Verifies the data integrity of the CDF file from its checksum.

VIRTUAL_RECORD_DATA - Static variable in interface gsfc.nssdc.cdf.CDFConstants

Z

zENTRY - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_DATA - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_DATASPEC - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_DATATYPE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_EXISTENCE - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_NAME - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zENTRY_NUMELEMS - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zMODEoff - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zMODEon1 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zMODEon2 - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_ALLOCATEBLOCK - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_ALLOCATEDFROM_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_ALLOCATEDDTO_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_ALLOCATERECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_BLOCKINGFACTOR_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_CACHESIZE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_COMPRESSION_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DATASPEC_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DATATYPE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DIMCOUNTS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DIMINDICES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DIMINTERVALS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DIMSIZES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_DIMVARYS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_EXISTENCE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_HYPERDATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_INITIALRECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_MAXallocREC_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_MAXREC_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVAR_NAME_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_nINDEXENTRIES_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_nINDEXLEVELS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_nINDEXRECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_NUMallocRECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_NUMBER_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_NUMDIMS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_NUMELEMS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_NUMRECS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_PADVALUE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RECCOUNT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RECINTERVAL_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RECNUMBER_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RECVARY_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_RESERVEPERCENT_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_SEQDATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_SEQPOS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_SPARSEARRAYS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVAR_SPARSERECORDS_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVARs_CACHESIZE_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVARs_MAXREC_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
zVARs_RECDATA_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants

zVARs_RECNUMBER_ - Static variable in interface gsfc.nssdc.cdf.CDFConstants
How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Overview

The Overview page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain four categories:

- Interfaces (italic)
- Classes
- Enums
- Exceptions
- Errors
- Annotation Types

Class/Interface

Each class, interface, nested class and nested interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description
Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Annotation Type

Each annotation type has its own separate page with the following sections:

- Annotation Type declaration
- Annotation Type description
- Required Element Summary
- Optional Element Summary
- Element Detail

Enum

Each enum has its own separate page with the following sections:

- Enum declaration
- Enum description
- Enum Constant Summary
- Enum Constant Detail

Tree (Class Hierarchy)

There is a [Class Hierarchy](#) page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all
packages.

- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

## Deprecated API

The [Deprecated API](http://localhost:8080/cdfdocs/help-doc.html#deprecated) page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

## Index

The [Index](http://localhost:8080/cdfdocs/help-doc.html#index) contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

## Prev/Next

These links take you to the next or previous class, interface, package, or related page.

## Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

## Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

## Constant Field Values

The [Constant Field Values](http://localhost:8080/cdfdocs/help-doc.html#constant-field-values) page lists the static final fields and their values.

*This help file applies to API documentation generated using the standard doclet.*
All Classes

Attribute
CDF
CDFConstants
CDFData
CDFDelegate
CDFException
CDFNativeLibrary
CDFObject
CDFTools
CDFTT2000
CDFUtils
Entry
Epoch
Epoch16
EpochNative
Variable
Hierarchy For Package gsfc.nssdc.cdf

Package Hierarchies:

All Packages

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.CDF (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDOBject)
  - gsfc.nssdc.cdf.CDFData (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDOBject)
  - gsfc.nssdc.cdf.CDFNativeLibrary (implements gsfc.nssdc.cdf.CDFDelegate)
  - gsfc.nssdc.cdf.CDFTools (implements gsfc.nssdc.cdf.CDFConstants)
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - gsfc.nssdc.cdf.CDFException (implements gsfc.nssdc.cdf.CDFConstants)

Interface Hierarchy

- gsfc.nssdc.cdf.CDFConstants
- gsfc.nssdc.cdf.CDFDelegate
- gsfc.nssdc.cdf.CDOBject
<table>
<thead>
<tr>
<th>Overview</th>
<th>Package</th>
<th>Class</th>
<th>Tree</th>
<th>Deprecated</th>
<th>Index</th>
<th>Help</th>
<th>PREV</th>
<th>NEXT</th>
<th>FRAMES</th>
<th>NO FRAMES</th>
<th>All Classes</th>
</tr>
</thead>
</table>
Hierarchy For Package gsfc.nssdc.cdf.util

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.util.CDFTT2000 (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.CDFUtils (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.Epoch (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.Epoch16 (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.EpochNative
public interface CDFConstants

This class defines the constants used by the CDF library and CDF Java APIs, and it mimics the cdf.h include file from the cdf distribution.

Version:
1.0

Field Summary

<table>
<thead>
<tr>
<th>static long</th>
<th>AHUFF_COMPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long</td>
<td>ALPHAOSF1_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAOSF1_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSd_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSd_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSg_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSg_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSi_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSi_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_EXISTS</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXgENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXrENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXzENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NAME_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NAME_TRUNC</td>
</tr>
<tr>
<td>static long</td>
<td>ATTRNUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMgENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMrENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMzENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
<tr>
<td>static long</td>
<td>....</td>
</tr>
</tbody>
</table>

- `ATTR_SCOPE_`
- `BACKWARD_
- `BACKWARDFILEoff`
- `BACKWARDFILEon`
- `BAD_ALLOCATE_RECS`
- `BAD_ARGUMENT`
- `BAD_ATTR_NAME`
- `BAD_ATTR_NUM`
- `BAD_BLOCKING_FACTOR`
- `BAD_CACHE_SIZE`
- `BAD_CDF_EXTENSION`
- `BAD_CDF_ID`
- `BAD_CDF_NAME`
- `BAD_CDFSTATUS`
- `BAD_CHECKSUM`
- `BAD_COMPRESSION_PARM`
<p>| static long | BAD_DATA_TYPE |
| static long | BAD_DECODING |
| static long | BAD_DIM_COUNT |
| static long | BAD_DIM_INDEX |
| static long | BAD_DIM_INTERVAL |
| static long | BAD_DIM_SIZE |
| static long | BAD_ENCODING |
| static long | BAD_ENTRY_NUM |
| static long | BAD_FNC_OR_ITEM |
| static long | BAD_FORMAT |
| static long | BAD_INITIAL_RECS |
| static long | BAD_MAJORORITY |
| static long | BAD_MALLOC |
| static long | BAD_NEGtoPOSfp0_MODE |
| static long | BAD_NUM_DIMS |
| static long | BAD_NUM_ELEMS |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>static long</td>
<td>BAD_NUM_VARS</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_READONLY_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_COUNT</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_INTERVAL</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_NUM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SCOPE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SCRATCH_DIR</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SPARSEARRAYS_PARM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_VAR_NAME</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_VAR_NUM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_zMODE</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_ALLOCATE_RECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_CHANGE</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_COMPRESS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_COPY</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_SPARSEARRAYS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_SPARSE_RECORDS</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ATTR_NAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ATTR_NAME_LEN256</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_BYTE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CACHESIZE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CHAR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CHECKSUM</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CLOSE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COPYRIGHT</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COPYRIGHT_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CREATE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DELETE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DOUBLE</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ENCODING_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EPOCH</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EPOCH16</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EXISTS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_FLOAT</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_FORMAT_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INFO_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT1</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT2</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT4</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT8</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INTERNAL_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAJORITY_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAX_DIMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAX_PARMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MIN_DIMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NAME_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NAME_TRUNC</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NEGtoPOSfp0_MODE_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMATTRS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMgATTRS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMrVARS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMvATTRS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMzVARS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_OK</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_OPEN_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_PATHNAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_READ_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_READONLY_MODE_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_REAL4</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_REAL8</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_RELEASE__</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_SAVE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_SCRATCHDIR__</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_STATUS__</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_STATUSTEXT_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_TIME_TT2000</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_UCHAR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_UINT1</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_UINT2</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_UINT4</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_VAR_NAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_VAR_NAME_LEN256</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_VERSION__</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_WARN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_WRITE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_zMODE_</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>static long</td>
<td>CDFwithSTATS_</td>
</tr>
<tr>
<td>static long</td>
<td>CHECKSUM_</td>
</tr>
<tr>
<td>static long</td>
<td>CHECKSUM_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CHECKSUM_NOT_ALLOWED</td>
</tr>
<tr>
<td>static long</td>
<td>CLOSE_</td>
</tr>
<tr>
<td>static long</td>
<td>COLUMN_MAJOR</td>
</tr>
<tr>
<td>static long</td>
<td>COMPRESS_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>COMPRESSION_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CONFIRM_</td>
</tr>
<tr>
<td>static long</td>
<td>CORRUPTED_V2_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>CORRUPTED_V3_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>CREATE_</td>
</tr>
<tr>
<td>static long</td>
<td>CURgENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>CURrENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>CURzENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>DATATYPE_MISMATCH</td>
</tr>
<tr>
<td>static long</td>
<td>DATATYPE_SIZE</td>
</tr>
<tr>
<td>static long</td>
<td>DECOMPRESSION_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>DECSTATION_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>DECSTATION_ENCODING</td>
</tr>
<tr>
<td>static byte</td>
<td>DEFAULT_BYTE_PADVALUE</td>
</tr>
<tr>
<td>static char</td>
<td>DEFAULT_CHAR_PADVALUE</td>
</tr>
<tr>
<td>static double</td>
<td>DEFAULT_DOUBLE_PADVALUE</td>
</tr>
<tr>
<td>static double</td>
<td>DEFAULT_EPOCH_PADVALUE</td>
</tr>
<tr>
<td>static float</td>
<td>DEFAULT_FLOAT_PADVALUE</td>
</tr>
<tr>
<td>static byte</td>
<td>DEFAULT_INT1_PADVALUE</td>
</tr>
<tr>
<td>static short</td>
<td>DEFAULT_INT2_PADVALUE</td>
</tr>
<tr>
<td>static int</td>
<td>DEFAULT_INT4_PADVALUE</td>
</tr>
<tr>
<td>static long</td>
<td>DEFAULT_INT8_PADVALUE</td>
</tr>
<tr>
<td>static float</td>
<td>DEFAULT_REAL4_PADVALUE</td>
</tr>
<tr>
<td>static double</td>
<td>DEFAULT_REAL8_PADVALUE</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>DEFAULT_TT2000_PADVALUE</td>
</tr>
<tr>
<td>static char</td>
<td>DEFAULT_UCHAR_PADVALUE</td>
</tr>
<tr>
<td>static short</td>
<td>DEFAULT_UINT1_PADVALUE</td>
</tr>
<tr>
<td>static int</td>
<td>DEFAULT_UINT2_PADVALUE</td>
</tr>
<tr>
<td>static long</td>
<td>DEFAULT_UINT4_PADVALUE</td>
</tr>
<tr>
<td>static long</td>
<td>DELETE_</td>
</tr>
<tr>
<td>static long</td>
<td>DID_NOT_COMPRESS</td>
</tr>
<tr>
<td>static long</td>
<td>DUMMY_TT2000_VALUE</td>
</tr>
<tr>
<td>static long</td>
<td>EMPTY_COMPRESSED_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>END_OF_VAR</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH1_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH1_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH2_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH2_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH3_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH3_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH4_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCH4_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCHx_FORMAT_MAX</td>
</tr>
<tr>
<td>static long</td>
<td>EPOCHx_STRING_MAX</td>
</tr>
<tr>
<td>static long</td>
<td>FORCED_PARAMETER</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_DATA_</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_DATASPEC_</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_DATATYPE_</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>gENTRY_NUMELEMS_</td>
</tr>
<tr>
<td>static long</td>
<td>GET_</td>
</tr>
<tr>
<td>static long</td>
<td>GETCDFCHECKSUM_</td>
</tr>
<tr>
<td>static long</td>
<td>GETCDFFILEBACKWARD_</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>GETCDFVALIDATE</td>
</tr>
<tr>
<td>static long</td>
<td>GETLEAPSECONDSENVVAR</td>
</tr>
<tr>
<td>static long</td>
<td>GLOBAL_SCOPE</td>
</tr>
<tr>
<td>static long</td>
<td>GZIP_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>HOST_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>HOST_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>HP_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>HP_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>HUFF_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>IBM_PC_OVERFLOW</td>
</tr>
<tr>
<td>static long</td>
<td>IBMPC_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>IBMPC_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>IBMRS_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>IBMRS_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_EPOCH_FIELD</td>
</tr>
<tr>
<td>static double</td>
<td>ILLEGAL_EPOCH_VALUE</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_FOR_SCOPE</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_IN_zMODE</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_ON_V1_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_TT2000_VALUE</td>
</tr>
<tr>
<td>static long</td>
<td>IS_A_NETCDF</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_COPYRIGHT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_INCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_RELEASE_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_subINCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_VERSION_</td>
</tr>
<tr>
<td>static long</td>
<td>MAC_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>MAC_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>MD5_CHECKSUM</td>
</tr>
<tr>
<td>static long</td>
<td>MULTI_FILE</td>
</tr>
<tr>
<td>static long</td>
<td>MULTI_FILE_FORMAT</td>
</tr>
<tr>
<td>static long</td>
<td>NA_FOR_VARIABLE</td>
</tr>
<tr>
<td>static long</td>
<td>NEGATIVE_FP_ZERO</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>static long</td>
<td>NEGtoPOSfp0off</td>
</tr>
<tr>
<td>static long</td>
<td>NEGtoPOSfp0on</td>
</tr>
<tr>
<td>static long</td>
<td>NETWORK_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>NETWORK_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>NeXT_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>NeXT_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>NO_ATTR_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_CDF_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_CHECKSUM</td>
</tr>
<tr>
<td>static long</td>
<td>NO_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>NO_DELETE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_ENTRY_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_MORE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_PADVALUE_SPECIFIED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SPARSEARRAYS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SPARSERECORDS</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>static long</td>
<td>NO_STATUS_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_ATTR</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_ENTRY</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_RECORD</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_VAR</td>
</tr>
<tr>
<td>static long</td>
<td>NO_VAR_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_VARS_IN_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NO_WRITE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NONE_CHECKSUM</td>
</tr>
<tr>
<td>static long</td>
<td>NOT_A_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NOT_A_CDF_OR_NOT_SUPPORTED</td>
</tr>
<tr>
<td>static long</td>
<td>NOVARY</td>
</tr>
<tr>
<td>static long</td>
<td>NULL_</td>
</tr>
<tr>
<td>static long</td>
<td>OPEN_</td>
</tr>
<tr>
<td>static long</td>
<td>OPTIMAL_ENCODING_TREES</td>
</tr>
<tr>
<td>static long</td>
<td>OTHER_CHECKSUM</td>
</tr>
<tr>
<td>static long</td>
<td>PAD_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>PPC_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>PPC_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>PRECEEDING_RECORDS_ALLOCATED</td>
</tr>
<tr>
<td>static long</td>
<td>PREV_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>PUT</td>
</tr>
<tr>
<td>static long</td>
<td>READ_ONLY_DISTRIBUTION</td>
</tr>
<tr>
<td>static long</td>
<td>READ_ONLY_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>READONLYoff</td>
</tr>
<tr>
<td>static long</td>
<td>READONLYon</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_DATA</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_DATASPEC</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_DATATYPE</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_NAME_</td>
</tr>
<tr>
<td>static long</td>
<td>rENTRY_NUMELEMS_</td>
</tr>
<tr>
<td>static long</td>
<td>RLE_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>RLE_OF_ZEROs</td>
</tr>
<tr>
<td>static long</td>
<td>ROW_MAJOR</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_ALLOCATEBLOCK_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_ALLOCATEDFROM_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_ALLOCATEDTO_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARALLOCATERECS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_BLOCKINGFACTOR_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_COMPRESSION_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATA_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATASPEC_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATATYPE__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DIMVARYS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_EXISTENCE__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_HYPERDATA__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_INITIALRECS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_MAXallocREC__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_MAXREC__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NAME__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXENTRIES__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXLEVELS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXRECORDS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMallocRECS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMBER__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMELEMS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMRECS__</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_PADVALUE__</td>
</tr>
<tr>
<td>Static long</td>
<td>Symbol</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>rVAR_RECORDS</td>
</tr>
<tr>
<td></td>
<td>rVAR_RECVARY</td>
</tr>
<tr>
<td></td>
<td>rVAR_reservepercent</td>
</tr>
<tr>
<td></td>
<td>rVAR_SEQDATA</td>
</tr>
<tr>
<td></td>
<td>rVAR_SEQPOS</td>
</tr>
<tr>
<td></td>
<td>rVAR_SPARSEARRAYS</td>
</tr>
<tr>
<td></td>
<td>rVAR_SPARSERECORDS</td>
</tr>
<tr>
<td></td>
<td>rVARs_CACHESIZE</td>
</tr>
<tr>
<td></td>
<td>rVARs_DIMCOUNTS</td>
</tr>
<tr>
<td></td>
<td>rVARs_DIMINDICES</td>
</tr>
<tr>
<td></td>
<td>rVARs_DIMINTERVALS</td>
</tr>
<tr>
<td></td>
<td>rVARs_DIMSIZES</td>
</tr>
<tr>
<td></td>
<td>rVARs_MAXREC</td>
</tr>
<tr>
<td></td>
<td>rVARs_NUMDIMS</td>
</tr>
<tr>
<td></td>
<td>rVARs_RECCOUNT</td>
</tr>
<tr>
<td></td>
<td>rVARs_RECDATA</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECINTERVAL_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECNUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>SAVE_</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_CREATE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_DELETE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_READ_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_WRITE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SELECT_</td>
</tr>
<tr>
<td>static long</td>
<td>SGi_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>SGi_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>SINGLE_FILE</td>
</tr>
<tr>
<td>static long</td>
<td>SINGLE_FILE_FORMAT</td>
</tr>
<tr>
<td>static long</td>
<td>SOME_ALREADY_ALLOCATED</td>
</tr>
<tr>
<td>static long</td>
<td>STAGE_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>STATUS_TEXT_</td>
</tr>
<tr>
<td>static long</td>
<td>SUN_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>SUN_ENCODING</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>TOO_MANY_PARMS</td>
</tr>
<tr>
<td>static long</td>
<td>TOO_MANY_VARS</td>
</tr>
<tr>
<td>static long</td>
<td>TT2000_0_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>TT2000_1_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>TT2000_2_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>TT2000_3_STRING_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>TT2000_TIME_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>UNKNOWN_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>UNKNOWN_SPARSENESS</td>
</tr>
<tr>
<td>static long</td>
<td>UNSUPPORTED_OPERATION</td>
</tr>
<tr>
<td>static long</td>
<td>VALIDATE_</td>
</tr>
<tr>
<td>static long</td>
<td>VALIDATEFILEoff</td>
</tr>
<tr>
<td>static long</td>
<td>VALIDATEFILEon</td>
</tr>
<tr>
<td>static long</td>
<td>VARAlreadyCLOSED</td>
</tr>
<tr>
<td>static long</td>
<td>VAR_CLOSE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>VAR_CREATE_ERROR</td>
<td></td>
</tr>
<tr>
<td>VAR_DELETE_ERROR</td>
<td></td>
</tr>
<tr>
<td>VAR_EXISTS</td>
<td></td>
</tr>
<tr>
<td>VAR_NAME_TRUNC</td>
<td></td>
</tr>
<tr>
<td>VAR_OPEN_ERROR</td>
<td></td>
</tr>
<tr>
<td>VAR_READ_ERROR</td>
<td></td>
</tr>
<tr>
<td>VAR_SAVE_ERROR</td>
<td></td>
</tr>
<tr>
<td>VAR_WRITE_ERROR</td>
<td></td>
</tr>
<tr>
<td>VARIABLE_SCOPE</td>
<td></td>
</tr>
<tr>
<td>VARY</td>
<td></td>
</tr>
<tr>
<td>VAX_DECODING</td>
<td></td>
</tr>
<tr>
<td>VAX_ENCODING</td>
<td></td>
</tr>
<tr>
<td>VIRTUAL_RECORD_DATA</td>
<td></td>
</tr>
<tr>
<td>zENTRY__</td>
<td></td>
</tr>
<tr>
<td>zENTRY_DATA__</td>
<td></td>
</tr>
<tr>
<td>zENTRY_DATASPEC__</td>
<td></td>
</tr>
<tr>
<td>Static Long</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>zENTRY_DATATYPE</td>
<td></td>
</tr>
<tr>
<td>zENTRY_EXISTENCE</td>
<td></td>
</tr>
<tr>
<td>zENTRY_NAME</td>
<td></td>
</tr>
<tr>
<td>zENTRY_NUMELEMS</td>
<td></td>
</tr>
<tr>
<td>zMODEoff</td>
<td></td>
</tr>
<tr>
<td>zMODEon1</td>
<td></td>
</tr>
<tr>
<td>zMODEon2</td>
<td></td>
</tr>
<tr>
<td>zVAR</td>
<td></td>
</tr>
<tr>
<td>zVAR_ALLOCATEBLOCK</td>
<td></td>
</tr>
<tr>
<td>zVAR_ALLOCATEDFROM</td>
<td></td>
</tr>
<tr>
<td>zVAR_ALLOCATEDTO</td>
<td></td>
</tr>
<tr>
<td>zVAR_ALLOCATERECS</td>
<td></td>
</tr>
<tr>
<td>zVAR_BLOCKINGFACTOR</td>
<td></td>
</tr>
<tr>
<td>zVAR_CACHESIZE</td>
<td></td>
</tr>
<tr>
<td>zVAR_COMPRESSION</td>
<td></td>
</tr>
<tr>
<td>zVAR_DATA</td>
<td></td>
</tr>
<tr>
<td>Static Long</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>zVAR_DATASPEC_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DATATYPE_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DIMCOUNTS_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DIMINDICES_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DIMINTERVALS_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DIMSIZES_</td>
<td></td>
</tr>
<tr>
<td>zVAR_DIMVARYS_</td>
<td></td>
</tr>
<tr>
<td>zVAR_EXISTENCE_</td>
<td></td>
</tr>
<tr>
<td>zVAR_HYPERDATA_</td>
<td></td>
</tr>
<tr>
<td>zVAR_INITIALRECS_</td>
<td></td>
</tr>
<tr>
<td>zVAR_MAXALLOCREC_</td>
<td></td>
</tr>
<tr>
<td>zVAR_MAXREC_</td>
<td></td>
</tr>
<tr>
<td>zVAR_NAME_</td>
<td></td>
</tr>
<tr>
<td>zVAR_nINDEXENTRIES_</td>
<td></td>
</tr>
<tr>
<td>zVAR_nINDEXLEVELS_</td>
<td></td>
</tr>
<tr>
<td>zVAR_nINDEXRECORDS_</td>
<td></td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMallocRECS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMDIMS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMELEMS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMRECS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_PADVALUE_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECCOUNT_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECINTERVAL_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECNUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECVARY_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RESERVEPERCENT_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SEQDATA_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SEQPOS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SPARSEARRAYS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SPARSERECORDS_</td>
</tr>
</tbody>
</table>
### Field Detail

#### CDF_MIN_DIMS

Static final long `CDF_MIN_DIMS`

See Also:
Constant Field Values

#### CDF_MAX_DIMS

Static final long `CDF_MAX_DIMS`

See Also:
Constant Field Values

#### CDF_VAR_NAME_LEN

Static final long `CDF_VAR_NAME_LEN`

See Also:
Constant Field Values

<table>
<thead>
<tr>
<th>static long</th>
<th>zVARs_CACHESIZE_</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long</td>
<td>zVARs_MAXREC_</td>
</tr>
<tr>
<td>static long</td>
<td>zVARs_RECDATA_</td>
</tr>
<tr>
<td>static long</td>
<td>zVARs_RECNUMBER_</td>
</tr>
</tbody>
</table>
CDF_VAR_NAME_LEN256

static final long CDF_VAR_NAME_LEN256

See Also:
   Constant Field Values

CDF_ATTRIB_NAME_LEN

static final long CDF_ATTRIB_NAME_LEN

See Also:
   Constant Field Values

CDF_ATTRIB_NAME_LEN256

static final long CDF_ATTRIB_NAME_LEN256

See Also:
   Constant Field Values

CDF_COPYRIGHT_LEN

static final long CDF_COPYRIGHT_LEN

See Also:
   Constant Field Values

CDF_STATUSTEXT_LEN
static final long CDF_STATUSTEXT_LEN

See Also:
   Constant Field Values

CDF_PATHNAME_LEN

static final long CDF_PATHNAME_LEN

See Also:
   Constant Field Values

EPOCH_STRING_LEN

static final long EPOCH_STRING_LEN

See Also:
   Constant Field Values

EPOCH1_STRING_LEN

static final long EPOCH1_STRING_LEN

See Also:
   Constant Field Values

EPOCH2_STRING_LEN

static final long EPOCH2_STRING_LEN

See Also:
EPOCH3_STRING_LEN

static final long EPOCH3_STRING_LEN

See Also:
Constant Field Values

EPOCH4_STRING_LEN

static final long EPOCH4_STRING_LEN

See Also:
Constant Field Values

EPOCHx_STRING_MAX

static final long EPOCHx_STRING_MAX

See Also:
Constant Field Values

EPOCHx_FORMAT_MAX

static final long EPOCHx_FORMAT_MAX

See Also:
Constant Field Values
EPOCH_STRING_LEN_EXTEND

static final long EPOCH_STRING_LEN_EXTEND

See Also:
   Constant Field Values

EPOCH1_STRING_LEN_EXTEND

static final long EPOCH1_STRING_LEN_EXTEND

See Also:
   Constant Field Values

EPOCH2_STRING_LEN_EXTEND

static final long EPOCH2_STRING_LEN_EXTEND

See Also:
   Constant Field Values

EPOCH3_STRING_LEN_EXTEND

static final long EPOCH3_STRING_LEN_EXTEND

See Also:
   Constant Field Values

EPOCH4_STRING_LEN_EXTEND

static final long EPOCH4_STRING_LEN_EXTEND
See Also:
   Constant Field Values

---

**TT2000_0_STRING_LEN**

static final long **TT2000_0_STRING_LEN**

See Also:
   Constant Field Values

---

**TT2000_1_STRING_LEN**

static final long **TT2000_1_STRING_LEN**

See Also:
   Constant Field Values

---

**TT2000_2_STRING_LEN**

static final long **TT2000_2_STRING_LEN**

See Also:
   Constant Field Values

---

**TT2000_3_STRING_LEN**

static final long **TT2000_3_STRING_LEN**

See Also:
   Constant Field Values
CDF_INT1

static final long CDF_INT1

See Also:
Constant Field Values

CDF_INT2

static final long CDF_INT2

See Also:
Constant Field Values

CDF_INT4

static final long CDF_INT4

See Also:
Constant Field Values

CDF_INT8

static final long CDF_INT8

See Also:
Constant Field Values

CDF_UINT1
static final long **CDF_UINT1**

See Also:

[Constant Field Values](#)

---

**CDF_UINT2**

static final long **CDF_UINT2**

See Also:

[Constant Field Values](#)

---

**CDF_UINT4**

static final long **CDF_UINT4**

See Also:

[Constant Field Values](#)

---

**CDF_REAL4**

static final long **CDF_REAL4**

See Also:

[Constant Field Values](#)

---

**CDF_REAL8**

static final long **CDF_REAL8**

See Also:

[Constant Field Values](#)
CDFConstants

Constant Field Values

CDF_EPOCH

static final long CDF_EPOCH

See Also:
Constant Field Values

CDF_EPOCH16

static final long CDF_EPOCH16

See Also:
Constant Field Values

CDF_TIME_TT2000

static final long CDF_TIME_TT2000

See Also:
Constant Field Values

CDF_BYTE

static final long CDF_BYTE

See Also:
Constant Field Values
CDF_FLOAT

static final long CDF_FLOAT

See Also:
   Constant Field Values

CDF_DOUBLE

static final long CDF_DOUBLE

See Also:
   Constant Field Values

CDF_CHAR

static final long CDF_CHAR

See Also:
   Constant Field Values

CDF_UCHAR

static final long CDF_UCHAR

See Also:
   Constant Field Values

NETWORK_ENCODING

static final long NETWORK_ENCODING
See Also:
   Constant Field Values

---

SUN_ENCODING

static final long SUN_ENCODING

See Also:
   Constant Field Values

---

VAX_ENCODING

static final long VAX_ENCODING

See Also:
   Constant Field Values

---

DECSTATION_ENCODING

static final long DECSTATION_ENCODING

See Also:
   Constant Field Values

---

SGi_ENCODING

static final long SGi_ENCODING

See Also:
   Constant Field Values
**IBMPC_ENCODING**

static final long `IBMPC_ENCODING`

See Also:

Constant Field Values

---

**IBMRS_ENCODING**

static final long `IBMRS_ENCODING`

See Also:

Constant Field Values

---

**HOST_ENCODING**

static final long `HOST_ENCODING`

See Also:

Constant Field Values

---

**PPC_ENCODING**

static final long `PPC_ENCODING`

See Also:

Constant Field Values

---

**HP_ENCODING**
static final long HP_ENCODING

See Also:
Constant Field Values

NeXT_ENCODING

static final long NeXT_ENCODING

See Also:
Constant Field Values

ALPHAOSF1_ENCODING

static final long ALPHAOSF1_ENCODING

See Also:
Constant Field Values

ALPHAVMSd_ENCODING

static final long ALPHAVMSd_ENCODING

See Also:
Constant Field Values

ALPHAVMSg_ENCODING

static final long ALPHAVMSg_ENCODING
**ALPHAVMSi_ENCODING**

static final long ALPHAVMSi_ENCODING

See Also:
Constant Field Values

**NETWORK_DECODING**

static final long NETWORK_DECODING

See Also:
Constant Field Values

**SUN_DECODING**

static final long SUN_DECODING

See Also:
Constant Field Values

**VAX_DECODING**

static final long VAX_DECODING

See Also:
Constant Field Values
DECSTATION_DECODING

static final long DECSTATION_DECODING

See Also:

Constant Field Values

---

SGi_DECODING

static final long SGi_DECODING

See Also:

Constant Field Values

---

IBMPC_DECODING

static final long IBMPC_DECODING

See Also:

Constant Field Values

---

IBMRS_DECODING

static final long IBMRS_DECODING

See Also:

Constant Field Values

---

HOST_DECODING
static final long HOST_DECODING

See Also:
Constant Field Values

PPC_DECODING

static final long PPC_DECODING

See Also:
Constant Field Values

MAC_ENCODING

static final long MAC_ENCODING

See Also:
Constant Field Values

MAC_DECODING

static final long MAC_DECODING

See Also:
Constant Field Values

HP_DECODING

static final long HP_DECODING

See Also:
**NeXT_DECODING**

static final long NeXT_DECODING

See Also:

Constant Field Values

**ALPHAOSF1_DECODING**

static final long ALPHAOSF1_DECODING

See Also:

Constant Field Values

**ALPHAVMSd_DECODING**

static final long ALPHAVMSd_DECODING

See Also:

Constant Field Values

**ALPHAVMSg_DECODING**

static final long ALPHAVMSg_DECODING

See Also:

Constant Field Values
ALPHAVMSi_DECODING

static final long ALPHAVMSi_DECODING

See Also:
- Constant Field Values

VARY

static final long VARY

See Also:
- Constant Field Values

NOVARY

static final long NOVARY

See Also:
- Constant Field Values

ROW_MAJOR

static final long ROW_MAJOR

See Also:
- Constant Field Values

COLUMN_MAJOR

static final long COLUMN_MAJOR
See Also:

Constant Field Values

---

**SINGLE_FILE**

static final long **SINGLE_FILE**

See Also:

Constant Field Values

---

**MULTI_FILE**

static final long **MULTI_FILE**

See Also:

Constant Field Values

---

**GLOBAL_SCOPE**

static final long **GLOBAL_SCOPE**

See Also:

Constant Field Values

---

**VARIABLE_SCOPE**

static final long **VARIABLE_SCOPE**

See Also:

Constant Field Values
READONLYon

static final long READONLYon

See Also:
Constant Field Values

READONLYoff

static final long READMEoff

See Also:
Constant Field Values

zMODEoff

static final long zMODEoff

See Also:
Constant Field Values

zMODEon1

static final long zMODEon1

See Also:
Constant Field Values

zMODEon2
static final long zMODEon2

See Also:
Constant Field Values

---

NEGtoPOSfp0on

static final long NEGtoPOSfp0on

See Also:
Constant Field Values

---

NEGtoPOSfp0off

static final long NEGtoPOSfp0off

See Also:
Constant Field Values

---

BACKWARDFILEon

static final long BACKWARDFILEon

See Also:
Constant Field Values

---

BACKWARDFILEoff

static final long BACKWARDFILEoff

See Also:
VALIDATEFILEOn

static final long VALIDATEFILEOn

See Also:
   Constant Field Values

VALIDATEFILEoff

static final long VALIDATEFILEoff

See Also:
   Constant Field Values

NO_CHECKSUM

static final long NO_CHECKSUM

See Also:
   Constant Field Values

NONE_CHECKSUM

static final long NONE_CHECKSUM

See Also:
   Constant Field Values
MD5_CHECKSUM

static final long MD5_CHECKSUM

See Also:
Constant Field Values

OTHER_CHECKSUM

static final long OTHER_CHECKSUM

See Also:
Constant Field Values

CDF_MAX_PARMS

static final long CDF_MAX_PARMS

See Also:
Constant Field Values

NO_COMPRESSION

static final long NO_COMPRESSION

See Also:
Constant Field Values

RLE_COMPRESSION

static final long RLE_COMPRESSION
See Also:

Constant Field Values

---

**HUFF_COMPRESSION**

static final long **HUFF_COMPRESSION**

See Also:

Constant Field Values

---

**AHUFF_COMPRESSION**

static final long **AHUFF_COMPRESSION**

See Also:

Constant Field Values

---

**GZIP_COMPRESSION**

static final long **GZIP_COMPRESSION**

See Also:

Constant Field Values

---

**RLE_OF_ZEROS**

static final long **RLE_OF_ZEROS**

See Also:

Constant Field Values
OPTIMAL_ENCODING_TREES

static final long OPTIMAL_ENCODING_TREES

See Also:
   Constant Field Values

NO_SPARSEARRAYS

static final long NO_SPARSEARRAYS

See Also:
   Constant Field Values

NO_SPARSERECORDS

static final long NO_SPARSERECORDS

See Also:
   Constant Field Values

PAD_SPARSERECORDS

static final long PAD_SPARSERECORDS

See Also:
   Constant Field Values

PREV_SPARSERECORDS
static final long PREV_SPARSERECORDS

See Also:
Constant Field Values

DEFAULT_BYTE_PADVALUE

static final byte DEFAULT_BYTE_PADVALUE

See Also:
Constant Field Values

DEFAULT_INT1_PADVALUE

static final byte DEFAULT_INT1_PADVALUE

See Also:
Constant Field Values

DEFAULT_UINT1_PADVALUE

static final short DEFAULT_UINT1_PADVALUE

See Also:
Constant Field Values

DEFAULT_INT2_PADVALUE

static final short DEFAULT_INT2_PADVALUE
CDFConstants

See Also:
  Constant Field Values

DEFAULT_UINT2_PADVALUE

static final int DEFAULT_UINT2_PADVALUE

See Also:
  Constant Field Values

DEFAULT_INT4_PADVALUE

static final int DEFAULT_INT4_PADVALUE

See Also:
  Constant Field Values

DEFAULT_UINT4_PADVALUE

static final long DEFAULT_UINT4_PADVALUE

See Also:
  Constant Field Values

DEFAULT_INT8_PADVALUE

static final long DEFAULT_INT8_PADVALUE

See Also:
  Constant Field Values
DEFAULT_REAL4_PADVALUE

static final float DEFAULT_REAL4_PADVALUE

See Also:
    Constant Field Values

DEFAULT_FLOAT_PADVALUE

static final float DEFAULT_FLOAT_PADVALUE

See Also:
    Constant Field Values

DEFAULT_REAL8_PADVALUE

static final double DEFAULT_REAL8_PADVALUE

See Also:
    Constant Field Values

DEFAULT_DOUBLE_PADVALUE

static final double DEFAULT_DOUBLE_PADVALUE

See Also:
    Constant Field Values

DEFAULT_CHAR_PADVALUE
static final char DEFAULT_CHAR_PADVALUE

See Also:

Constant Field Values

---

DEFAULT_UCHAR_PADVALUE

static final char DEFAULT_UCHAR_PADVALUE

See Also:

Constant Field Values

---

DEFAULT_EPOCH_PADVALUE

static final double DEFAULT_EPOCH_PADVALUE

See Also:

Constant Field Values

---

DEFAULT_TT2000_PADVALUE

static final long DEFAULT_TT2000_PADVALUE

See Also:

Constant Field Values

---

ILLEGAL_EPOCH_VALUE

static final double ILLEGAL_EPOCH_VALUE

See Also:
ILLEGAL_TT2000_VALUE

static final long ILLEGAL_TT2000_VALUE

See Also:
Constant Field Values

DUMMY_TT2000_VALUE

static final long DUMMY_TT2000_VALUE

See Also:
Constant Field Values

VIRTUAL_RECORD_DATA

static final long VIRTUAL_RECORD_DATA

See Also:
Constant Field Values

DID_NOT_COMPRESS

static final long DID_NOT_COMPRESS

See Also:
Constant Field Values
VAR_ALREADY_CLOSED

static final long VAR_ALREADY_CLOSED

See Also:
Constant Field Values

SINGLE_FILE_FORMAT

static final long SINGLE_FILE_FORMAT

See Also:
Constant Field Values

NO_PADVALUE_SPECIFIED

static final long NO_PADVALUE_SPECIFIED

See Also:
Constant Field Values

NO_VARS_IN_CDF

static final long NO_VARS_IN_CDF

See Also:
Constant Field Values

MULTI_FILE_FORMAT

static final long MULTI_FILE_FORMAT
See Also:
  Constant Field Values

SOME_ALREADY_ALLOCATED

static final long SOME_ALREADY_ALLOCATED

See Also:
  Constant Field Values

PRECEEDING_RECORDS_ALLOCATED

static final long PRECEEDING_RECORDS_ALLOCATED

See Also:
  Constant Field Values

CDF_OK

static final long CDF_OK

See Also:
  Constant Field Values

ATTR_NAME_TRUNC

static final long ATTR_NAME_TRUNC

See Also:
  Constant Field Values
CDF_NAME_TRUNC

static final long CDF_NAME_TRUNC

See Also:
Constant Field Values

VAR_NAME_TRUNC

static final long VAR_NAME_TRUNC

See Also:
Constant Field Values

NEGATIVE_FP_ZERO

static final long NEGATIVE_FP_ZERO

See Also:
Constant Field Values

FORCED_PARAMETER

static final long FORCED_PARAMETER

See Also:
Constant Field Values

NA_FOR_VARIABLE
static final long NA_FOR_VARIABLE

See Also:
   Constant Field Values

CDF_WARN

static final long CDF_WARN

See Also:
   Constant Field Values

ATTR_EXISTS

static final long ATTR_EXISTS

See Also:
   Constant Field Values

BAD_CDF_ID

static final long BAD_CDF_ID

See Also:
   Constant Field Values

BAD_DATA_TYPE

static final long BAD_DATA_TYPE

See Also:
BAD_DIM_SIZE

static final long BAD_DIM_SIZE

See Also:
Constant Field Values

BAD_DIM_INDEX

static final long BAD_DIM_INDEX

See Also:
Constant Field Values

BAD_ENCODING

static final long BAD_ENCODING

See Also:
Constant Field Values

BAD_MAJORORITY

static final long BAD_MAJORORITY

See Also:
Constant Field Values
BAD_NUM_DIMS

static final long BAD_NUM_DIMS

See Also:
Constant Field Values

BAD_REC_NUM

static final long BAD_REC_NUM

See Also:
Constant Field Values

BAD_SCOPE

static final long BAD_SCOPE

See Also:
Constant Field Values

BAD_NUM_ELEMS

static final long BAD_NUM_ELEMS

See Also:
Constant Field Values

CDF_OPEN_ERROR

static final long CDF_OPEN_ERROR
See Also:
   Constant Field Values

CDF_EXISTS

static final long CDF_EXISTS

See Also:
   Constant Field Values

BAD_FORMAT

static final long BAD_FORMAT

See Also:
   Constant Field Values

BAD_ALLOCATE_RECS

static final long BAD_ALLOCATE_RECS

See Also:
   Constant Field Values

BAD_CDF_EXTENSION

static final long BAD_CDF_EXTENSION

See Also:
   Constant Field Values
NO_SUCH_ATTR

static final long NO_SUCH_ATTR

See Also:
Constant Field Values

NO_SUCH_ENTRY

static final long NO_SUCH_ENTRY

See Also:
Constant Field Values

NO_SUCH_VAR

static final long NO_SUCH_VAR

See Also:
Constant Field Values

VAR_READ_ERROR

static final long VAR_READ_ERROR

See Also:
Constant Field Values

VAR_WRITE_ERROR
static final long VAR_WRITE_ERROR

See Also:
Constant Field Values

BAD_ARGUMENT

static final long BAD_ARGUMENT

See Also:
Constant Field Values

IBM_PC_OVERFLOW

static final long IBM_PC_OVERFLOW

See Also:
Constant Field Values

TOO_MANY_VARS

static final long TOO_MANY_VARS

See Also:
Constant Field Values

VAR_EXISTS

static final long VAR_EXISTS
See Also:

Constant Field Values

BAD_MALLOC

static final long BAD_MALLOC

See Also:

Constant Field Values

NOT_A_CDF

static final long NOT_A_CDF

See Also:

Constant Field Values

CORRUPTED_V2_CDF

static final long CORRUPTED_V2_CDF

See Also:

Constant Field Values

VAR_OPEN_ERROR

static final long VAR_OPEN_ERROR

See Also:

Constant Field Values
BAD_INITIAL_RECS

static final long BAD_INITIAL_RECS

See Also:
Constant Field Values

BAD_BLOCKING_FACTOR

static final long BAD_BLOCKING_FACTOR

See Also:
Constant Field Values

END_OF_VAR

static final long END_OF_VAR

See Also:
Constant Field Values

BAD_CDFSTATUS

static final long BAD_CDFSTATUS

See Also:
Constant Field Values

CDF_INTERNAL_ERROR
static final long CDF_INTERNAL_ERROR

See Also:
   Constant Field Values

BAD_NUM_VARS

static final long BAD_NUM_VARS

See Also:
   Constant Field Values

BAD_REC_COUNT

static final long BAD_REC_COUNT

See Also:
   Constant Field Values

BAD_REC_INTERVAL

static final long BAD_REC_INTERVAL

See Also:
   Constant Field Values

BAD_DIM_COUNT

static final long BAD_DIM_COUNT

See Also:
**BAD_DIM_INTERVAL**

static final long BAD_DIM_INTERVAL

See Also:
Constant Field Values

**BAD_VAR_NUM**

static final long BAD_VAR_NUM

See Also:
Constant Field Values

**BAD_ATTR_NUM**

static final long BAD_ATTR_NUM

See Also:
Constant Field Values

**BAD_ENTRY_NUM**

static final long BAD_ENTRY_NUM

See Also:
Constant Field Values
BAD_ATTR_NAME

static final long BAD_ATTR_NAME

See Also:
    Constant Field Values

BAD_VAR_NAME

static final long BAD_VAR_NAME

See Also:
    Constant Field Values

NO_ATTR_SELECTED

static final long NO_ATTR_SELECTED

See Also:
    Constant Field Values

NO_ENTRY_SELECTED

static final long NO_ENTRY_SELECTED

See Also:
    Constant Field Values

NO_VAR_SELECTED

static final long NO_VAR_SELECTED
BAD_CDF_NAME
static final long BAD_CDF_NAME

CANNOT_CHANGE
static final long CANNOT_CHANGE

NO_STATUS_SELECTED
static final long NO_STATUS_SELECTED

NO_CDF_SELECTED
static final long NO_CDF_SELECTED

See Also:
Constant Field Values
READ_ONLY_DISTRIBUTION

static final long READ_ONLY_DISTRIBUTION

See Also:
   Constant Field Values

CDF_CLOSE_ERROR

static final long CDF_CLOSE_ERROR

See Also:
   Constant Field Values

VAR_CLOSE_ERROR

static final long VAR_CLOSE_ERROR

See Also:
   Constant Field Values

BAD_FNC_OR_ITEM

static final long BAD_FNC_OR_ITEM

See Also:
   Constant Field Values

ILLEGAL_ON_V1_CDF
static final long ILLEGAL_ON_V1_CDF

See Also:
   Constant Field Values

BAD_CACHE_SIZE

static final long BAD_CACHE_SIZE

See Also:
   Constant Field Values

CDF_CREATE_ERROR

static final long CDF_CREATE_ERROR

See Also:
   Constant Field Values

NO_SUCH_CDF

static final long NO_SUCH_CDF

See Also:
   Constant Field Values

VAR_CREATE_ERROR

static final long VAR_CREATE_ERROR

See Also:
Constant Field Values

READ_ONLY_MODE

static final long READ_ONLY_MODE

See Also:
Constant Field Values

ILLEGAL_IN_zMODE

static final long ILLEGAL_IN_zMODE

See Also:
Constant Field Values

BAD_zMODE

static final long BAD_zMODE

See Also:
Constant Field Values

BAD_READONLY_MODE

static final long BAD_READONLY_MODE

See Also:
Constant Field Values
CDF_READ_ERROR

static final long CDF_READ_ERROR

See Also:
Constant Field Values

CDF_WRITE_ERROR

static final long CDF_WRITE_ERROR

See Also:
Constant Field Values

ILLEGAL_FOR_SCOPE

static final long ILLEGAL_FOR_SCOPE

See Also:
Constant Field Values

NO_MORE_ACCESS

static final long NO_MORE_ACCESS

See Also:
Constant Field Values

BAD_DECODING

static final long BAD_DECODING
BAD_NEGtoPOSfp0_MODE

static final long BAD_NEGtoPOSfp0_MODE

See Also:
 Constant Field Values

UNSupported_OPERATION

static final long UNSUPPORTED_OPERATION

See Also:
 Constant Field Values

CDF_SAVE_ERROR

static final long CDF_SAVE_ERROR

See Also:
 Constant Field Values

VAR_SAVE_ERROR

static final long VAR_SAVE_ERROR

See Also:
 Constant Field Values
**NO_WRITE_ACCESS**

static final long NO_WRITE_ACCESS

See Also:  
Constant Field Values

**NO_DELETE_ACCESS**

static final long NO_DELETE_ACCESS

See Also:  
Constant Field Values

**CDF_DELETE_ERROR**

static final long CDF_DELETE_ERROR

See Also:  
Constant Field Values

**VAR_DELETE_ERROR**

static final long VAR_DELETE_ERROR

See Also:  
Constant Field Values

**UNKNOWN_COMPRESSION**
static final long UNKNOWN_COMPRESSION

See Also:
Constant Field Values

CANNOT_COMPRESS

static final long CANNOT_COMPRESS

See Also:
Constant Field Values

DECOMPRESSSION_ERROR

static final long DECOMPRESSSION_ERROR

See Also:
Constant Field Values

COMPRESSION_ERROR

static final long COMPRESSION_ERROR

See Also:
Constant Field Values

EMPTY_COMPRESSED_CDF

static final long EMPTY_COMPRESSED_CDF
See Also:
Constant Field Values

BAD_COMPRESSION_PARM

static final long BAD_COMPRESSION_PARM

See Also:
Constant Field Values

UNKNOWN_SPARSENESSE

static final long UNKNOWN_SPARSENESSE

See Also:
Constant Field Values

CANNOT_SPARSERECORDS

static final long CANNOT_SPARSERECORDS

See Also:
Constant Field Values

CANNOT_SPARSEARRAYS

static final long CANNOT_SPARSEARRAYS

See Also:
Constant Field Values
TOO_MANY_PARMS

static final long TOO_MANY_PARMS

See Also:
   Constant Field Values

NO_SUCH_RECORD

static final long NO_SUCH_RECORD

See Also:
   Constant Field Values

CANNOT_ALLOCATE_RECORDS

static final long CANNOT_ALLOCATE_RECORDS

See Also:
   Constant Field Values

CANNOT_COPY

static final long CANNOT_COPY

See Also:
   Constant Field Values

SCRATCH_DELETE_ERROR
static final long SCRATCH_DELETE_ERROR

See Also:
Constant Field Values

---

SCRATCH_CREATE_ERROR

static final long SCRATCH_CREATE_ERROR

See Also:
Constant Field Values

---

SCRATCH_READ_ERROR

static final long SCRATCH_READ_ERROR

See Also:
Constant Field Values

---

SCRATCH_WRITE_ERROR

static final long SCRATCH_WRITE_ERROR

See Also:
Constant Field Values

---

BAD_SPARSEARRAYS_PARM

static final long BAD_SPARSEARRAYS_PARM

See Also:
<table>
<thead>
<tr>
<th>Constant Field Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BAD_SCRATCH_DIR</strong></td>
</tr>
<tr>
<td>static final long BAD_SCRATCH_DIR</td>
</tr>
<tr>
<td>See Also: Constant Field Values</td>
</tr>
<tr>
<td><strong>DATATYPE_MISMATCH</strong></td>
</tr>
<tr>
<td>static final long DATATYPE_MISMATCH</td>
</tr>
<tr>
<td>See Also: Constant Field Values</td>
</tr>
<tr>
<td><strong>NOT_A_CDF_OR_NOT_SUPPORTED</strong></td>
</tr>
<tr>
<td>static final long NOT_A_CDF_OR_NOT_SUPPORTED</td>
</tr>
<tr>
<td>See Also: Constant Field Values</td>
</tr>
<tr>
<td><strong>CORRUPTED_V3_CDF</strong></td>
</tr>
<tr>
<td>static final long CORRUPTED_V3_CDF</td>
</tr>
<tr>
<td>See Also: Constant Field Values</td>
</tr>
</tbody>
</table>
ILLEGAL_EPOCH_FIELD

static final long ILLEGAL_EPOCH_FIELD

See Also:
Constant Field Values

BAD_CHECKSUM

static final long BAD_CHECKSUM

See Also:
Constant Field Values

CHECKSUM_ERROR

static final long CHECKSUM_ERROR

See Also:
Constant Field Values

CHECKSUM_NOT_ALLOWED

static final long CHECKSUM_NOT_ALLOWED

See Also:
Constant Field Values

IS_A_NETCDF

static final long IS_A_NETCDF
See Also:
  Constant Field Values

---

**TT2000_TIME_ERROR**

static final long `TT2000_TIME_ERROR`

See Also:
  Constant Field Values

---

**CREATE**

static final long `CREATE`

See Also:
  Constant Field Values

---

**OPEN**

static final long `OPEN`

See Also:
  Constant Field Values

---

**DELETE**

static final long `DELETE`

See Also:
  Constant Field Values
CLOSE_

static final long CLOSE_

See Also:
  Constant Field Values

SELECT_

static final long SELECT_

See Also:
  Constant Field Values

CONFIRM_

static final long CONFIRM_

See Also:
  Constant Field Values

GET_

static final long GET_

See Also:
  Constant Field Values

PUT_
static final long PUT_

See Also:
Constant Field Values

---

SAVE_

static final long SAVE_

See Also:
Constant Field Values

---

BACKWARD_

static final long BACKWARD_

See Also:
Constant Field Values

---

GETCDFFILEBACKWARD_

static final long GETCDFFILEBACKWARD_

See Also:
Constant Field Values

---

CHECKSUM_

static final long CHECKSUM_

See Also:
CDFConstants

### GETCDFCHECKSUM_

`static final long GETCDFCHECKSUM_`

**See Also:**

Constant Field Values

### VALIDATE_

`static final long VALIDATE_`

**See Also:**

Constant Field Values

### GETCDFVALIDATE_

`static final long GETCDFVALIDATE_`

**See Also:**

Constant Field Values

### GETLEAPSECONDSENVVAR_

`static final long GETLEAPSECONDSENVVAR_`

**See Also:**

Constant Field Values
NULL_

static final long NULL_

See Also:
Constant Field Values

CDF_

static final long CDF_

See Also:
Constant Field Values

CDF_NAME_

static final long CDF_NAME_

See Also:
Constant Field Values

CDF_ENCODING_

static final long CDF_ENCODING_

See Also:
Constant Field Values

CDF_DECODING_

static final long CDF_DECODING_
See Also:

Constant Field Values

---

CDF_MAJORITY_

static final long CDF_MAJORITY_

See Also:

Constant Field Values

---

CDF_FORMAT_

static final long CDF_FORMAT_

See Also:

Constant Field Values

---

CDF_COPYRIGHT_

static final long CDF_COPYRIGHT_

See Also:

Constant Field Values

---

CDF_NUMrVARS_

static final long CDF_NUMrVARS_

See Also:

Constant Field Values
CDF_NUMzVARS_

static final long CDF_NUMzVARS_

See Also:
Constant Field Values

CDF_NUMATTRS_

static final long CDF_NUMATTRS_

See Also:
Constant Field Values

CDF_NUMgATTRS_

static final long CDF_NUMgATTRS_

See Also:
Constant Field Values

CDF_NUMvATTRS_

static final long CDF_NUMvATTRS_

See Also:
Constant Field Values

CDF_VERSION_
static final long CDF_VERSION_

See Also:
Constant Field Values

CDF_RELEASE_

static final long CDF_RELEASE_

See Also:
Constant Field Values

CDF_INCREMENT_

static final long CDF_INCREMENT_

See Also:
Constant Field Values

CDF_STATUS_

static final long CDF_STATUS_

See Also:
Constant Field Values

CDF_READONLY_MODE_

static final long CDF_READONLY_MODE_
CDFConstants

See Also:  
Constant Field Values

---

**CDF_zMODE**

static final long **CDF_zMODE**

See Also:  
Constant Field Values

---

**CDF_NEGtoPOSfp0_MODE**

static final long **CDF_NEGtoPOSfp0_MODE**

See Also:  
Constant Field Values

---

**LIB_COPYRIGHT**

static final long **LIB_COPYRIGHT**

See Also:  
Constant Field Values

---

**LIB_VERSION**

static final long **LIB_VERSION**

See Also:  
Constant Field Values
**LIB_RELEASE**

static final long LIB_RELEASE

See Also:
- Constant Field Values

**LIB_INCREMENT**

static final long LIB_INCREMENT

See Also:
- Constant Field Values

**LIB_subINCREMENT**

static final long LIB_subINCREMENT

See Also:
- Constant Field Values

**rVARs_NUMDIMS**

static final long rVARs_NUMDIMS

See Also:
- Constant Field Values

**rVARs_DIMSIZES**
static final long rVARs_DIMSIZEs_

See Also:
Constant Field Values

rVARs_MAXREC_

static final long rVARs_MAXREC_

See Also:
Constant Field Values

rVARs_RECDATA_

static final long rVARs_RECDATA_

See Also:
Constant Field Values

rVARs_RECNUMBER_

static final long rVARs_RECNUMBER_

See Also:
Constant Field Values

rVARs_RECCOUNT_

static final long rVARs_RECCOUNT_

See Also:
rVARs_RECINTERVAL_

static final long rVARs_RECINTERVAL_

See Also:
Constant Field Values

rVARs_DIMINDICES_

static final long rVARs_DIMINDICES_

See Also:
Constant Field Values

rVARs_DIMCOUNTS_

static final long rVARs_DIMCOUNTS_

See Also:
Constant Field Values

rVARs_DIMINTERVALS_

static final long rVARs_DIMINTERVALS_

See Also:
Constant Field Values
static final long rVAR_

See Also:
Constant Field Values

static final long rVAR_NAME_

See Also:
Constant Field Values

static final long rVAR_DATATYPE_

See Also:
Constant Field Values

static final long rVAR_NUMELEMENTS_

See Also:
Constant Field Values

static final long rVAR_RECVARY_
rVAR_DIMVARYS_
static final long rVAR_DIMVARYS_

See Also:
Constant Field Values

rVAR_NUMBER_
static final long rVAR_NUMBER_

See Also:
Constant Field Values

rVAR_DATA_
static final long rVAR_DATA_

See Also:
Constant Field Values

rVAR_HYPERDATA_
static final long rVAR_HYPERDATA_

See Also:
Constant Field Values
static final long rVAR_SEQDATA_

See Also:
Constant Field Values

static final long rVAR_SEQPOS_

See Also:
Constant Field Values

static final long rVAR_MAXREC_

See Also:
Constant Field Values

static final long rVAR_MAXallocREC_

See Also:
Constant Field Values

rVAR_DATASPEC_
static final long rVAR_DATASPEC_

See Also:
   Constant Field Values

rVAR_PADVALUE_

static final long rVAR_PADVALUE_

See Also:
   Constant Field Values

rVAR INITIALRECS_

static final long rVAR_INITIALRECS_

See Also:
   Constant Field Values

rVAR BLOCKINGFACTOR_

static final long rVAR_BLOCKINGFACTOR_

See Also:
   Constant Field Values

rVAR nINDEXRECORDS_

static final long rVAR_nINDEXRECORDS_

See Also:
**rVAR_nINDEXENTRIES_**

static final long `rVAR_nINDEXENTRIES_`

See Also:  
Constant Field Values

**rVAR_EXISTENCE_**

static final long `rVAR_EXISTENCE_`

See Also:  
Constant Field Values

**zVARs_MAXREC_**

static final long `zVARs_MAXREC_`

See Also:  
Constant Field Values

**zVARs_RECDATA_**

static final long `zVARs_RECDATA_`

See Also:  
Constant Field Values
zVAR_

static final long zVAR_

See Also:
Constant Field Values

zVAR_NAME_

static final long zVAR_NAME_

See Also:
Constant Field Values

zVAR_DATATYPE_

static final long zVAR_DATATYPE_

See Also:
Constant Field Values

zVAR_NUMELEMS_

static final long zVAR_NUMELEMS_

See Also:
Constant Field Values

zVAR_NUMDIMS_

static final long zVAR_NUMDIMS_
See Also:
Constant Field Values

---

**zVAR_DIMSIZES**

static final long zVAR_DIMSIZES

See Also:
Constant Field Values

---

**zVAR_RECVARY**

static final long zVAR_RECVARY

See Also:
Constant Field Values

---

**zVAR_DIMVARYS**

static final long zVAR_DIMVARYS

See Also:
Constant Field Values

---

**zVAR_NUMBER**

static final long zVAR_NUMBER

See Also:
Constant Field Values
zVAR_DATA_

static final long zVAR_DATA_

See Also:
   Constant Field Values

zVAR_HYPERDATA_

static final long zVAR_HYPERDATA_

See Also:
   Constant Field Values

zVAR_SEQDATA_

static final long zVAR_SEQDATA_

See Also:
   Constant Field Values

zVAR_SEQPOS_

static final long zVAR_SEQPOS_

See Also:
   Constant Field Values

zVAR_MAXREC_
static final long zVAR_MAXREC_

See Also:
   Constant Field Values

zVAR_MAXallocREC_
static final long zVAR_MAXallocREC_

See Also:
   Constant Field Values

zVAR_DATASPEC_
static final long zVAR_DATASPEC_

See Also:
   Constant Field Values

zVAR_PADVALUE_
static final long zVAR_PADVALUE_

See Also:
   Constant Field Values

zVAR_INITIALRECS_
static final long zVAR_INITIALRECS_
CDFConstants

See Also:
   Constant Field Values

---

**zVAR_BLOCKINGFACTOR**

static final long **zVAR_BLOCKINGFACTOR**

See Also:
   Constant Field Values

---

**zVAR_nINDEXRECORDS**

static final long **zVAR_nINDEXRECORDS**

See Also:
   Constant Field Values

---

**zVAR_nINDEXENTRIES**

static final long **zVAR_nINDEXENTRIES**

See Also:
   Constant Field Values

---

**zVAR_EXISTENCE**

static final long **zVAR_EXISTENCE**

See Also:
   Constant Field Values
static final long zVAR_RECNUMBER_ 

See Also: 
Constant Field Values

static final long zVAR_RECCOUNT_ 

See Also: 
Constant Field Values

static final long zVAR_RECINTERVAL_ 

See Also: 
Constant Field Values

static final long zVAR_DIMINDICES_ 

See Also: 
Constant Field Values

static final long zVAR_DIMCOUNTS_
static final long zVAR_DIMCOUNTS_

See Also:
    Constant Field Values

zVAR_DIMINTERVALS_

static final long zVAR_DIMINTERVALS_

See Also:
    Constant Field Values

ATTR_

static final long ATTR_

See Also:
    Constant Field Values

ATTR_SCOPE_

static final long ATTR_SCOPE_

See Also:
    Constant Field Values

ATTR_NAME_

static final long ATTR_NAME_

See Also:
**ATTR_NUMBER_**

static final long **ATTR_NUMBER_**

See Also:

Constant Field Values

**ATTR_MAXgENTRY_**

static final long **ATTR_MAXgENTRY_**

See Also:

Constant Field Values

**ATTR_NUMgENTRIES_**

static final long **ATTR_NUMgENTRIES_**

See Also:

Constant Field Values

**ATTR_MAXrENTRY_**

static final long **ATTR_MAXrENTRY_**

See Also:

Constant Field Values
ATTR_NUMrENTRIES_

static final long ATTR_NUMrENTRIES_

See Also:
   Constant Field Values

ATTR_MAXzENTRY_

static final long ATTR_MAXzENTRY_

See Also:
   Constant Field Values

ATTR_NUMzENTRIES_

static final long ATTR_NUMzENTRIES_

See Also:
   Constant Field Values

ATTR_EXISTENCE_

static final long ATTR_EXISTENCE_

See Also:
   Constant Field Values

gENTRY_

static final long gENTRY_
See Also:
   Constant Field Values

---

**gENTRY_EXISTENCE_**

static final long **gENTRY_EXISTENCE_**

See Also:
   Constant Field Values

---

**gENTRY_DATATYPE_**

static final long **gENTRY_DATATYPE_**

See Also:
   Constant Field Values

---

**gENTRY_NUMELEMS_**

static final long **gENTRY_NUMELEMS_**

See Also:
   Constant Field Values

---

**gENTRY_DATASPEC_**

static final long **gENTRY_DATASPEC_**

See Also:
   Constant Field Values
gENTRY_DATA_

static final long gENTRY_DATA_

See Also:
Constant Field Values

rENTRY_

static final long rENTRY_

See Also:
Constant Field Values

rENTRY_NAME_

static final long rENTRY_NAME_

See Also:
Constant Field Values

rENTRY_EXISTENCE_

static final long rENTRY_EXISTENCE_

See Also:
Constant Field Values

rENTRY_DATATYPE_
static final long rENTRY_DATATYPE_

See Also:
Constant Field Values

rENTRY_NUMELEMS_

static final long rENTRY_NUMELEMS_

See Also:
Constant Field Values

rENTRY_DATASPEC_

static final long rENTRY_DATASPEC_

See Also:
Constant Field Values

rENTRY_DATA_

static final long rENTRY_DATA_

See Also:
Constant Field Values

zENTRY_

static final long zENTRY_

See Also:
Constant Field Values
CDFConstants

Constant Field Values

zENTRY_NAME_

static final long zENTRY_NAME_

See Also:
Constant Field Values

zENTRY_EXISTENCE_

static final long zENTRY_EXISTENCE_

See Also:
Constant Field Values

zENTRY_DATATYPE_

static final long zENTRY_DATATYPE_

See Also:
Constant Field Values

zENTRY_NUMELEMS_

static final long zENTRY_NUMELEMS_

See Also:
Constant Field Values
zENTRY_DATASPEC_

static final long zENTRY_DATASPEC_

See Also:

Constant Field Values

zENTRY_DATA_

static final long zENTRY_DATA_

See Also:

Constant Field Values

STATUS_TEXT_

static final long STATUS_TEXT_

See Also:

Constant Field Values

CDF_CACHESIZE_

static final long CDF_CACHESIZE_

See Also:

Constant Field Values

rVARs_CACHESIZE_

static final long rVARs_CACHESIZE_
See Also:

Constant Field Values

zVARs_CACHESIZE_

static final long zVARs_CACHESIZE_

See Also:

Constant Field Values

rVAR_CACHESIZE_

static final long rVAR_CACHESIZE_

See Also:

Constant Field Values

zVAR_CACHESIZE_

static final long zVAR_CACHESIZE_

See Also:

Constant Field Values

zVARs_RECNUMBER_

static final long zVARs_RECNUMBER_

See Also:

Constant Field Values
rVARALLOCATERECS_

static final long rVARALLOCATERECS_

See Also:
   Constant Field Values

zVARALLOCATERECS_

static final long zVARALLOCATERECS_

See Also:
   Constant Field Values

DATATYPE_SIZE_

static final long DATATYPE_SIZE_

See Also:
   Constant Field Values

CURgENTRY_EXISTENCE_

static final long CURgENTRY_EXISTENCE_

See Also:
   Constant Field Values

CURrENTRY_EXISTENCE_
static final long CURzENTRY_EXISTENCE_

See Also:
Constant Field Values

CURzENTRY_EXISTENCE_

static final long CURzENTRY_EXISTENCE_

See Also:
Constant Field Values

CDF_INFO_

static final long CDF_INFO_

See Also:
Constant Field Values

CDF_COMPRESSION_

static final long CDF_COMPRESSION_

See Also:
Constant Field Values

zVAR_COMPRESSION_

static final long zVAR_COMPRESSION_
See Also: Constant Field Values

zVAR_SPARSERECORDS_

static final long zVAR_SPARSERECORDS_

See Also: Constant Field Values

zVAR_SPARSEARRAYS_

static final long zVAR_SPARSEARRAYS_

See Also: Constant Field Values

zVAR_ALLOCATEBLOCK_

static final long zVAR_ALLOCATEBLOCK_

See Also: Constant Field Values

zVAR_NUMRECS_

static final long zVAR_NUMRECS_

See Also: Constant Field Values
zVAR_NUMallocRECS_

static final long zVAR_NUMallocRECS_

See Also:
Constant Field Values

rVAR_COMPRESSION_

static final long rVAR_COMPRESSION_

See Also:
Constant Field Values

rVAR_SPARSERECORDS_

static final long rVAR_SPARSERECORDS_

See Also:
Constant Field Values

rVAR_SPARSEARRAYS_

static final long rVAR_SPARSEARRAYS_

See Also:
Constant Field Values

rVAR_ALLOCATEBLOCK_
static final long rVAR_ALLOCATEBLOCK_

See Also:
Constant Field Values

rVAR_NUMRECS_

static final long rVAR_NUMRECS_

See Also:
Constant Field Values

rVAR_NUMallocRECS_

static final long rVAR_NUMallocRECS_

See Also:
Constant Field Values

rVAR_ALLOCATEDFROM_

static final long rVAR_ALLOCATEDFROM_

See Also:
Constant Field Values

rVAR_ALLOCATEDTO_

static final long rVAR_ALLOCATEDTO_

See Also:
**zVAR_ALLOCATEDFROM**

static final long zVAR_ALLOCATEDFROM

See Also:
Constant Field Values

**zVAR_ALLOCATEDTO**

static final long zVAR_ALLOCATEDTO

See Also:
Constant Field Values

**zVAR_nINDEXLEVELS**

static final long zVAR_nINDEXLEVELS

See Also:
Constant Field Values

**rVAR_nINDEXLEVELS**

static final long rVAR_nINDEXLEVELS

See Also:
Constant Field Values
CDF_SCRATCHDIR_

static final long CDF_SCRATCHDIR_

See Also:
   Constant Field Values

rVAR_RESERVEPERCENT_

static final long rVAR_RESERVEPERCENT_

See Also:
   Constant Field Values

zVAR_RESERVEPERCENT_

static final long zVAR_RESERVEPERCENT_

See Also:
   Constant Field Values

rVAR_RECORDS_

static final long rVAR_RECORDS_

See Also:
   Constant Field Values

zVAR_RECORDS_

static final long zVAR_RECORDS_
STAGE_CACHESIZE_

static final long STAGE_CACHESIZE_

See Also:
Constant Field Values

COMPRESS_CACHESIZE_

static final long COMPRESS_CACHESIZE_

See Also:
Constant Field Values

CDF_CHECKSUM_

static final long CDF_CHECKSUM_

See Also:
Constant Field Values

CDFwithSTATS_

static final long CDFwithSTATS_

See Also:
Constant Field Values
static final long CDF_ACCESS_

See Also:
Constant Field Values
gsfc.nssdc.cdf

Interface CDFObject

All Known Implementing Classes:
   Attribute, CDF, CDFData, Entry, Variable

public interface CDFObject

CDFObject provides the base interface for all CDF objects. CDF objects mean the CDF, Attribute, Entry and Variable objects. All these objects need to implement this interface.

Version:
   1.0

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>delete()</td>
<td>Deletes the current object.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
<td>Returns the name of the current object.</td>
</tr>
<tr>
<td>void</td>
<td>rename(java.lang.String name)</td>
<td>Renames the current object.</td>
</tr>
</tbody>
</table>

Method Detail

ggetName

getName
java.lang.String **getName()**

Returns the name of the current object.

**Returns:**
the name of the current object

---

**rename**

void **rename**(java.lang.String name) throws **CDFException**

Renames the current object.

**Parameters:**
named - the new object name

**Throws:**
* CDFException - if an error occurs renaming the current object

---

**delete**

void **delete**() throws **CDFException**

Deletes the current object.

**Throws:**
* CDFException - if an error occurs deleting the current object
public class CDF

extends java.lang.Object

implements CDFObject, CDFConstants

The CDF class is the main class used to interact with a CDF file.

Notes:

- All files are placed in zMODE 2 upon opening or creation
- Variable attributes are handled slightly differently from C.
  - Each variable has a java.util.Vector of attributes.
  - This vector contains only those vAttributes that have a z entry for this variable.
  - Therefore, the index for a given variable Attribute may not be the same for another variable.

<table>
<thead>
<tr>
<th>CDF dataType</th>
<th>Java dataType</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF_BYTE</td>
<td>java.lang.Byte</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>java.lang.Byte</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_UINT1</td>
<td>java.lang.Short</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>java.lang.Short</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_UINT2</td>
<td>java.lang.Integer</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT4</td>
<td>java.lang.Integer</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_UINT4</td>
<td>java.lang.Long</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT8</td>
<td>java.lang.Long</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_FLOAT</td>
<td>java.lang.Float</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_REAL4</td>
<td>java.lang.Float</td>
<td>Y/Y</td>
</tr>
</tbody>
</table>
Version:

1.0, 2.0 03/18/05 Selection of current attribute is done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called. Sync'd the CDF (id) for every JNI calls.

See Also:

Attribute, CDFException, Variable

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSq_DECODING, ALPHAVMSq_ENCODING, ALPHAVMSi_DECODING, ATTR__, ATTR_EXISTENCE__, ATTR_EXISTS, ATTR_MAXgENTRY__, ATTR_MAXrENTRY__, ATTR_MAXzENTRY__, ATTR_NAME__, ATTR_NAME_TRUNC, ATTR_NUMBER__, ATTR_NUMgENTRIES__, ATTR_NUMrENTRIES__, ATTR_NUMzENTRIES__, ATTR_SCOPE__, BACKWARD__, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJOR, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS, CANNOT_SPARECORDS, CDF__, CDF_ACCESS__, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE__, CDF_CHAR, CDF_CHECKSUM__, CDF_CLOSE_ERROR, CDF_COMPRESSION__, CDF_COPYRIGHT__, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING__, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING__, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT__, CDF_INCREMENT__, CDF_INFO__, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_MAJOR, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME__, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE__, CDF_NUMATTRS__, CDF_NUMgATTRS__
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void close()</td>
<td>Closes this CDF file.</td>
</tr>
<tr>
<td>long confirmCDFCacheSize()</td>
<td>Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.</td>
</tr>
<tr>
<td>long confirmCompressCacheSize()</td>
<td>Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).</td>
</tr>
<tr>
<td>long confirmDecoding()</td>
<td>Gets the CDF decoding method defined for this CDF.</td>
</tr>
<tr>
<td>long confirmNegtoPosfp0()</td>
<td>Gets the -0.0 to 0.0 translation flag set for this CDF.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>long confirmReadOnlyMode ()</td>
<td>Gets the value of the read-only mode flag set for this CDF file.</td>
</tr>
<tr>
<td>long confirmStageCacheSize ()</td>
<td>Gets the number of 512-byte cache buffers defined for the staging scratch file.</td>
</tr>
<tr>
<td>long confirmzMode ()</td>
<td>Gets the zMode set for this CDF.</td>
</tr>
<tr>
<td>static CDF create (java.lang.String path)</td>
<td>Creates a CDF file in the current directory.</td>
</tr>
<tr>
<td>static CDF create (java.lang.String path, int flag)</td>
<td>Deprecated. Use setFileBackward(long) method to set the file backward flag and create(String) to create file instead.</td>
</tr>
<tr>
<td>void delete ()</td>
<td>Deletes this CDF file.</td>
</tr>
<tr>
<td>void finalize ()</td>
<td>Do the necessary cleanup when garbage collector reaps it.</td>
</tr>
<tr>
<td>Attribute getAttribute (long attrNum)</td>
<td>Gets the attribute for the given attribute number.</td>
</tr>
<tr>
<td>Attribute getAttribute (java.lang.String attrName)</td>
<td>Gets the attribute for the given attribute name.</td>
</tr>
<tr>
<td>long getAttributeID (java.lang.String attrName)</td>
<td>Gets the id of the given attribute.</td>
</tr>
<tr>
<td>java.util.Vector getAttributes ()</td>
<td>Gets all the global and variable attributes defined for this CDF.</td>
</tr>
<tr>
<td>long getChecksum ()</td>
<td>Gets the checksum method, if any, applied to the CDF.</td>
</tr>
<tr>
<td>static long getChecksumEnvVar ()</td>
<td>Gets the value of the CDF_CHECKSUM environment variable.</td>
</tr>
<tr>
<td>java.lang.String getCompression ()</td>
<td>Gets the string representation of the compression type and parameters defined for this CDF.</td>
</tr>
<tr>
<td>long[] getCompressionParms ()</td>
<td>Gets the compression parameters set for this CDF.</td>
</tr>
<tr>
<td>long getCompressionPct ()</td>
<td>Gets the compression percentage set for this CDF.</td>
</tr>
<tr>
<td>long getCompressionType ()</td>
<td>Gets the compression type set for this CDF.</td>
</tr>
<tr>
<td>java.lang.String getCopyright ()</td>
<td>Gets the CDF copyright statement for this CDF.</td>
</tr>
<tr>
<td>Method Type</td>
<td>Method Name</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CDFDelegate</td>
<td>getDelegate()</td>
</tr>
<tr>
<td>long</td>
<td>getEncoding()</td>
</tr>
<tr>
<td>static boolean</td>
<td>getFileBackward()</td>
</tr>
<tr>
<td>static int</td>
<td>getFileBackwardEnvVar()</td>
</tr>
<tr>
<td>long</td>
<td>getFormat()</td>
</tr>
<tr>
<td>java.util.Vector</td>
<td>getGlobalAttributes()</td>
</tr>
<tr>
<td>long</td>
<td>getID()</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getLeapSecondsTableEnvVar()</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getLibraryCopyright()</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getLibraryVersion()</td>
</tr>
<tr>
<td>long</td>
<td>getMajority()</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
</tr>
<tr>
<td>long</td>
<td>getNumAttrs()</td>
</tr>
<tr>
<td>long</td>
<td>getNumGattrs()</td>
</tr>
<tr>
<td>long</td>
<td>getNumRvars()</td>
</tr>
<tr>
<td>long</td>
<td>getNumVars()</td>
</tr>
<tr>
<td>long</td>
<td>getNumVattrs()</td>
</tr>
<tr>
<td>long</td>
<td>getNumZvars()</td>
</tr>
<tr>
<td>Method/Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>java.util.Vector getOrphanAttributes()</code></td>
<td>Gets the variable attributes defined for this CDF that are not associated with any variables.</td>
</tr>
<tr>
<td><code>java.util.Vector getRecord(long recNum, long[] varIDs)</code></td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>java.util.Vector getRecord(long recNum, long[] varIDs, long[] status)</code></td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>java.util.Vector getRecord(long recNum, java.lang.String[] strVars)</code></td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>java.util.Vector getRecord(long recNum, java.lang.String[] strVars, long[] status)</code></td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>long getStatus()</code></td>
<td>Gets the status of the most recent CDF JNI/library function call.</td>
</tr>
<tr>
<td><code>static java.lang.String getStatusText(long statusCode)</code></td>
<td>Gets the status text of the most recent CDF JNI/library function call.</td>
</tr>
<tr>
<td><code>static boolean getValidate()</code></td>
<td>Gets the file validation mode.</td>
</tr>
<tr>
<td><code>Variable getVariable(long varNum)</code></td>
<td>Gets the variable object for the given variable number.</td>
</tr>
<tr>
<td><code>Variable getVariable(java.lang.String varName)</code></td>
<td>Gets the variable object for the given variable name.</td>
</tr>
<tr>
<td><code>java.util.Vector getVariableAttributes()</code></td>
<td>Gets the variable attributes defined for this CDF.</td>
</tr>
<tr>
<td><code>long getVariableID(java.lang.String varName)</code></td>
<td>Gets the ID of the given variable.</td>
</tr>
<tr>
<td><code>java.util.Vector getVariables()</code></td>
<td>Gets the z variables defined for this CDF.</td>
</tr>
<tr>
<td><code>java.lang.String getVersion()</code></td>
<td>Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).</td>
</tr>
<tr>
<td><code>static CDF open(java.lang.String path)</code></td>
<td>Open a CDF file for read/write, the default mode for opening a CDF.</td>
</tr>
<tr>
<td><code>static CDF open(java.lang.String path, long readOnly)</code></td>
<td>Open a CDF file.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>putRecord</code></td>
<td><code>putRecord(long recNum, long[] varIDs, java.util.Vector myData)</code>&lt;br&gt;Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>putRecord</code></td>
<td><code>putRecord(long recNum, long[] varIDs, java.util.Vector myData, long[] status)</code>&lt;br&gt;Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>putRecord</code></td>
<td><code>putRecord(long recNum, java.lang.String[] strVars, java.util.Vector myData)</code>&lt;br&gt;Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>putRecord</code></td>
<td><code>putRecord(long recNum, java.lang.String[] strVars, java.util.Vector myData, long[] status)</code>&lt;br&gt;Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td><code>rename</code></td>
<td><code>rename(java.lang.String path)</code>&lt;br&gt;Renames the current CDF.</td>
</tr>
<tr>
<td><code>save</code></td>
<td><code>save()</code>&lt;br&gt;Saves this CDF file without closing.</td>
</tr>
<tr>
<td><code>selectCDFCacheSize</code></td>
<td><code>selectCDFCacheSize(long cacheSize)</code>&lt;br&gt;Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF).</td>
</tr>
<tr>
<td><code>selectCompressCacheSize</code></td>
<td><code>selectCompressCacheSize(long compressCacheSize)</code>&lt;br&gt;Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF).</td>
</tr>
<tr>
<td><code>selectDecoding</code></td>
<td><code>selectDecoding(long decoding)</code>&lt;br&gt;Defines the CDF decoding method to be used for this CDF.</td>
</tr>
<tr>
<td><code>selectNegtoPosfp0</code></td>
<td><code>selectNegtoPosfp0(long negtoPosfp0)</code>&lt;br&gt;Defines whether to translate -0.0 to 0.0 for reading or writing.</td>
</tr>
<tr>
<td><code>selectReadOnlyMode</code></td>
<td><code>selectReadOnlyMode(long readOnly)</code>&lt;br&gt;Sets the desired read-only mode.</td>
</tr>
<tr>
<td><code>selectStageCacheSize</code></td>
<td><code>selectStageCacheSize(long stageCacheSize)</code>&lt;br&gt;Sets the number of 512-byte cache buffers to be used for the staging scratch file (for the current CDF).</td>
</tr>
<tr>
<td><code>setChecksum</code></td>
<td><code>setChecksum(long checksum)</code>&lt;br&gt;Specifies the checksum option applied to the CDF.</td>
</tr>
<tr>
<td><code>setCompression</code></td>
<td><code>setCompression(long cType, long[] cParms)</code>&lt;br&gt;Sets the compression type and parameters for this CDF.</td>
</tr>
<tr>
<td><code>setDelegate</code></td>
<td><code>setDelegate(CDFDelegate delegate)</code>&lt;br&gt;This is a placeholder for future expansions/extensions.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>setEncoding</code>&lt;br&gt;<code>void setEncoding(long encoding)</code></td>
<td>Defines the encoding method to be used for this CDF.</td>
</tr>
<tr>
<td><code>setFileBackward</code>&lt;br&gt;<code>static void setFileBackward(long flag)</code></td>
<td>Sets the file backward flag so that when a new CDF file is created, it will be created in either the older V2.7 version or the current library version, i.e., V3.*.</td>
</tr>
<tr>
<td><code>setFormat</code>&lt;br&gt;<code>void setFormat(long format)</code></td>
<td>Specifies the format of this CDF.</td>
</tr>
<tr>
<td><code>setInfoWarningOff</code>&lt;br&gt;<code>void setInfoWarningOff()</code></td>
<td>Sets the informational (status code &gt; 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function off.</td>
</tr>
<tr>
<td><code>setInfoWarningOn</code>&lt;br&gt;<code>void setInfoWarningOn()</code></td>
<td>Sets the informational (status code &gt; 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function on.</td>
</tr>
<tr>
<td><code>setMajority</code>&lt;br&gt;<code>void setMajority(long majority)</code></td>
<td>Sets the variable majority for this CDF.</td>
</tr>
<tr>
<td><code>setValidate</code>&lt;br&gt;<code>static void setValidate(long mode)</code></td>
<td>Sets the file validation mode so that when a CDF file is open, it will be validated accordingly.</td>
</tr>
<tr>
<td><code>toString</code>&lt;br&gt;<code>java.lang.String toString()</code></td>
<td>Gets the name of this CDF.</td>
</tr>
<tr>
<td><code>verifyChecksum</code>&lt;br&gt;<code>long verifyChecksum()</code></td>
<td>Verifies the data integrity of the CDF file from its checksum.</td>
</tr>
</tbody>
</table>

**Methods inherited from class java.lang.Object**

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

**Method Detail**

**create**

`public static CDF create(java.lang.String path)`

Throws `CDFException`

Creates a CDF file in the current directory. By default, a single-file CDF is created and it's the preferred format. However, if the user wants to create a multi-file CDF, its file format needs to be changed as following:

```java
CDF cdf = null;
cdf = CDF.create("test");
cdf.setFormat(MULTI_FILE);
```
For the single-file format CDF, the above example would have created a single-file CDF called 'test.cdf'. See Chapter 1 of the CDF User's Guide for more information about the file format options. **Notes:**
The newly created file will be of the same version as the CDF library, as a V3.*. To create a backward file, i.e., V2.7, there are two options that can be used. Use the static method setFileBackward to set the backward flag. The following example will create backward file for test1.cdf and test2.cdf, but a V3.* file for test3.cdf.

```java
CDF cdf1, cdf2, cdf3;
CDF.setFileBackward(BACKWARDFILEon);
cdf1 = CDF.create("test1");
cdf2 = CDF.create("test2");
CDF.setFileBackward(BACKWARDFILEoff);
cdf3 = CDF.create("test3");
```

Alternatively, use an environment variable to control the backward file creation. The environment variable CDF_FILEBACKWARD on Unix or Windows or CDF$FILEBACKWARD on Open/VMS is used. When it is set to TRUE, a V2.7 file(s) will be created automatically. In the following example, both test1.cdf and test2.cdf will be V2.7 if environment variable CDF_FILEBACKWARD (or CDF$FILEBACKWARD) is TRUE.

```java
CDF cdf1 = CDF.create("test1");
CDF cdf2 = CDF.create("test2");
```

**Parameters:**
- **path** - the full pathname of the CDF file to be created

**Returns:**
- the newly created CDF file/object

**Throws:**
- **CDFException** - if there was a problem creating a CDF file

---

**create**

```java
public static CDF create(java.lang.String path,
                        int flag)
    throws CDFException
```

**Deprecated.** Use `setFileBackward(long)` method to set the file backward flag and `create(String)` to create file instead.

Creates a CDF file in the current directory. By default, a single-file CDF is created and it's the preferred format. The following example will create a CDF file:

```java
CDF cdf = null;
cdf = CDF.create("test", 0);
```
For the single-file format CDF, the above example would have created a single-file CDF called 'test.cdf'. The newly created file will be of the same version as the CDF library. To create a backward file, i.e., V2.7, use a different argument for the flag.

```java
CDF cdf;
cdf = CDF.create("test", 1);
```

**Parameters:**
- path - the full pathname of the CDF file to be created
- flag the file backward indicator flag. Passed 0 if a file of current library version is to be created. Not 0 if a backward is to be created.

**Returns:**
- the newly created CDF file/object

**Throws:**
- CDFException - if there was a problem creating a CDF file

---

### open

```java
public static CDF open(java.lang.String path)
  throws CDFException
```

Open a CDF file for read/write, the default mode for opening a CDF. If the user wants only to read the file, the file must be opened in read-only mode as following:

```java
CDF cdf = CDF.open(fileName, READONLYon);
```

Note: Opening a file with read/write mode will cause the checksum signature to be recomputed every time the file is closed. (NEW) Each open CDF file is subjected to a default data validating process (some overheads) that will perform sanity checks. To overwrite it, you can use one of two ways: 1) Use the CDF static method setValidate before calling the open method: CDF.setValidate (VALIDATEFILEoff); 2) Set the environment variable CDF_VALIDATE (CDF$VALIDATE on VMS) to no

**Parameters:**
- path - the full pathname of the CDF file to be opened

**Returns:**
- the CDF object that represents the CDF file the user requested for opening

**Throws:**
- CDFException - if there was a major problem opening a file Not always that a CDFException will be thrown. It is good practice to check the status from this open method to see if some others problems, e.g.,
invalid checksum is being detected, may occur. Use something like the followings

```java
if (cdf.getStatus() != CDF_OK)
{
    if (cdf.getStatus() == CHECKSUM_ERROR)
        .......
}
```

where cdf is the returned object from the open method. It is up to each individual to determine whether to continue to use a CDF file with an error like checksum.

---

**open**

```java
public static CDF open(java.lang.String path,
                       long readOnly)
    throws CDFException
```

Open a CDF file. A CDF file can be opened in read-only or read/write mode. If a file is opened in read-only mode, the user can only read values out of the file. Any operation other than reading data will throw a CDFException. If the user wants to modify the contents of a file, the file must be opened in read/write mode as following:

```java
CDF cdf = CDF.open(fileName, READONLYoff);
```

**Parameters:**

- **path** - the full pathname of the CDF file to be opened
- **readOnly** - read-only flag that should be one the following:
  - READONLYon - opens the file in read only mode.
  - READONLYoff - opens the file in read/write mode

**Returns:**

the CDF object that represents the CDF file the user requested for opening

**Throws:**

CDFException - if there was a major problem opening a file Not always that a CDFException will be thrown. It is good pratice to check the status from this open method to see if some others problems, e.g., invalid checksum is being detected, may occur. Use something like the followings

```java
if (cdf.getStatus() != CDF_OK)
{
    if (cdf.getStatus() == CHECKSUM_ERROR)
        .......
}
```
where cdf is the returned object from the open method. It is up to each individual to determine whether to continue to use a CDF file with an error like checksum.

---

**getLibraryVersion**

```java
public static java.lang.String getLibraryVersion()
    throws CDFException
```

Retrieve library version/release/increment/sub_increment information associated with the CDF library.

**Throws:**

- `CDFException` - If there was a problem retrieving the information associated with this CDF file

---

**getLibraryCopyright**

```java
public static java.lang.String getLibraryCopyright()
    throws CDFException
```

Retrieve library copyright information associated with the CDF library.

**Throws:**

- `CDFException` - If there was a problem retrieving the information associated with this CDF file

---

**close**

```java
public void close()
    throws CDFException
```

Closes this CDF file. It is essential that a CDF that has been created or modified by an application be closed before the program exits. If the CDF is not closed, the file will be corrupted and unreadable. This is because the cache buffers maintained by the CDF library will not have been written to the CDF file(s).

The following example closes a CDF file:

```
cdf.close();
```

**Throws:**

- `CDFException` - if there was a problem closing the CDF file
**getID**

```java
public long getID()
```

Gets the id of this CDF file.

**Returns:**

the id of this CDF file

---

**getEncoding**

```java
public long getEncoding()
```

Gets the encoding method defined for this CDF.

**Returns:**

The encoding method defined for this CDF file. One of the encoding methods described in the setEncoding method is returned.

---

**setEncoding**

```java
public void setEncoding(long encoding)
```

Throws: `CDFException`

Defines the encoding method to be used for this CDF. A CDF's data encoding affects how its attribute entry and variable data values are stored. By default, attribute entry and variable data values passed into the CDF library are always stored using the host machine's native encoding. For example, if a CDF file is created without specifying what encoding method should be should on a IBM PC, the IBMPC_ENCODING method is used. This method becomes useful if someone wants to create a CDF file that will be read on a machine that is different from the machine the CDF file was created. A CDF with any of the supported encodings may be read from and written to any supported computer. See section 2.2.8 of the CDF User's Guide for a detailed description of the encodings listed below.

**Parameters:**

- `encoding` - the encoding method to be used for this CDF that should be one of the following:
  - HOST_ENCODING
  - NETWORK_ENCODING
  - SUN_ENCODING
  - VAX_ENCODING
  - DECSTATION_ENCODING
  - SGi_ENCODING
  - IBMPC_ENCODING
selectDecoding

public void selectDecoding(long decoding)

throws CDFException

Defines the CDF decoding method to be used for this CDF. A CDF's decoding affects how its attribute entry and variable data values are passed out to a calling application. The decoding for a CDF may be selected any number of times while the CDF is open. Selecting a decoding does not affect how the values are store in the CDF file(s) - only how the values are decoded by the CDF library.

Parameters:

decoding - the decoding method to be used for this CDF that should be one of the following:

- HOST_DECODING - this is the default decoding
- NETWORK_DECODING
- SUN_DECODING
- VAX_DECODING
- DECSTATION_DECODING
- SGi_DECODING
- IBMPC_DECODING
- IBMRS_DECODING
- PPC_DECODING
- HP_DECODING
- NeXT_DECODING
- ALPHAOSF1_DECODING
- ALPHAVMSd_DECODING
- ALPHAVMSg_DECODING
- ALPHAVMSi_DECODING

Throws:

CDFException - if there was a problem selecting the requested decoding method
public long confirmDecoding() throws CDFException

Gets the CDF decoding method defined for this CDF.

Returns:
The decoding method set for this CDF file. One of the decoding methods defined in the selectDecoding method is returned.

Throws:
CDFException - if there was a problem getting the decoding method set for this CDF file

selectCDFCacheSize

public void selectCDFCacheSize(long cacheSize) throws CDFException

Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF). The concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:
cacheSize - the number of 512-byte cache buffers

Throws:
CDFException - if there was a problem setting the CDF cache size

confirmCDFCacheSize

public long confirmCDFCacheSize() throws CDFException

Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.

Returns:
the number of 512-byte cache buffers set for this CDF

Throws:
CDFException - if there was a problem getting the CDF cache size

selectNegtoPosfp0
public void selectNegtoPosfp0(long negtoPosfp0)
  throws CDFException

Defines whether to translate -0.0 to 0.0 for reading or writing. Negative floating-point zero (-0.0) is legal on
cOMPuters that use IEEE 754 floating-point representation (e.g. most UNIX-based computers and the PC) but is
illegal on VAXes and DEC alphas running OpenVMS operating system. If this mode disabled, a warning
(NEGATIVE_FP_ZERO) is returned when -0.0 is read from a CDF (and the decoding is that of a VAX or DEC
Alpha running OpenVMS) or written to a CDF (and the encoding is that of a VAX or DEC Alpha running i
OpenVMS).

Parameters:
    negtoPosfp0 - flag to translate -0.0 to 0.0 (NEGtoPOSfp0on = on, NEGtoPOSfp0off = off)

Throws:
    CDFException - if there was a problem setting the -0.0 to 0.0 translation flag

confirmNegtoPosfp0

public long confirmNegtoPosfp0()
  throws CDFException

Gets the -0.0 to 0.0 translation flag set for this CDF.

Returns:
    flag to translate -0.0 to 0.0 (NEGtoPOSfp0on = on, NEGtoPOSfp0off = off)

Throws:
    CDFException - if there was a problem getting the value of the -0.0 to 0.0 translation flag

getFormat

public long getFormat()

Gets the CDF format defined for this CDF.

Returns:
    the format of this CDF (SINGLE_FILE = single-file CDF, MULTI_FILE = multi-file CDF)

setFormat

public void setFormat(long format)
Specifies the format of this CDF. A CDF's format can't be changed once any variables are created. See section 1.4 of the CDF User's Guide for more detailed information about the file format options.

**Parameters:**

- `format` - the CDF file format to be used that should be one of the following:
  - SINGLE_FILE - This is the default. The CDF consists of only one file.
  - MULTI_FILE - The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

**Throws:**

- `CDFException` - if there was a problem setting a file format

---

### getVersion

**public java.lang.String getVersion()**

Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).

**Returns:**

- the CDF library version number that was used to create this CDF

---

### getMajority

**public long getMajority()**

Gets the variable majority defined for this CDF.

**Returns:**

- the variable majority defined for this CDF (ROW_MAJOR = row major, COLUMN_MAJOR = column major)

---

### setMajority

**public void setMajority(long majority) throws CDFException**

Sets the variable majority for this CDF. The variable majority of a CDF describes how variable values within each variable array (record) are stored. Each variable in a CDF has the same majority.
Parameters:

majority - The majority to be used in storing data (ROW_MAJOR = row major, COLUMN_MAJOR = column major)

Throws:

CDFException - if a problem occurred in setting a majority

---

**getNumAttrs**

class CDF:

public long `getNumAttrs()`

Gets the total number of global and variable attributes in this CDF.

Returns:

the total number of global and variable attributes in this CDF

---

**getNumGattrs**

class CDF:

public long `getNumGattrs()`

Gets the number of global attributes in this CDF.

Returns:

the number of global attributes in this CDF file

---

**getNumVattrs**

class CDF:

public long `getNumVattrs()`

Gets the number of variable attributes in this CDF. Since r variables are not supported by the CDF Java APIs, the number of z variables is always returned.

Returns:

the number of variable attributes in this CDF file

---

**getNumRvars**

class CDF:

public long `getNumRvars()`
Gets the number of r variables. Zero is returned since r variables are not supported. Z variables can do everything r variables can do plus more.

**Returns:**
the number of r variables in this CDF file

---

**getNumZvars**

```java
public long getNumZvars()
```

Gets the number of z variables in this CDF file.

**Returns:**
the number of z variables in this CDF file

---

**getCopyright**

```java
public java.lang.String getCopyright()
```

Gets the CDF copyright statement for this CDF.

**Returns:**
the CDF copyright statement

---

**selectReadOnlyMode**

```java
public void selectReadOnlyMode(long readOnly)
    throws CDFException
```

Sets the desired read-only mode. See the description of the read-only flag defined in the open method in this class for details. Caveat: Arbitrary changing the read-only mode to READONLYon while doing writing/updating will cause a problem to the file if the checksum bit is turned on (as the checksum signature may not get updated and a warning for data integrity will be issued when the file is open later).

**Parameters:**
readOnly - read-only flag (READONLYon = on, READONLYoff = off)

**Throws:**
CDFException - if a problem occurred in setting a flag
confirmReadOnlyMode

public long confirmReadOnlyMode() throws CDFException

    Gets the value of the read-only mode flag set for this CDF file.

    Returns:
    read-only flag (READONLYon = on, READONLYoff = off)

    Throws:
    CDFException - if a problem occurred in getting the value of the read-only flag set for this CDF file

getCompressionType

public long getCompressionType()  

    Gets the compression type set for this CDF.  

    Returns:
    the compression type set for this CDF - one of the following is returned:

        ■ NO_COMPRESSION - no compression  
        ■ RLE_COMPRESSION - Run-length compression  
        ■ HUFF_COMPRESSION - Huffman compression  
        ■ AHUFF_COMPRESSION - Adaptive Huffman compression  
        ■ GZIP_COMPRESSION - Gnu's "zip" compression

getCompressionPct

public long getCompressionPct()  

    Gets the compression percentage set for this CDF.  

    Returns:
    the compression percentage set for this CDF.

getCompressionParms

public long[] getCompressionParms()
 Gets the compression parameters set for this CDF. See the description of the setCompression method in this class for more information.

**Returns:**
the compression parameter set for this CDF

---

**setCompression**

```java
public void setCompression(long cType,
                          long[] cParms)
```

Sets the compression type and parameters for this CDF.

**Parameters:**
- `cType` - the compression type to be applied to this CDF that should be one of the following:
  - NO_COMPRESSION - no compression
  - RLE_COMPRESSION - Run-length compression. Currently, only the run-length encoding of zeros is supported. The compression parameter must be set to RLE_OF_ZEROS.
  - HUFF_COMPRESSION - Huffman compression. Currently, only optimal encoding trees are supported. The compression parameter must be set to OPTIMAL_ENCODING_TREES.
  - AHUFF_COMPRESSION - Adaptive Huffman compression. Currently, only optimal encoding trees are supported. The compression parameter must be set to OPTIMAL_ENCODING_TREES.
  - GZIP_COMPRESSION - Gnu's "zip" compression. The compression parameter may range from 1 to 9. 1 provides the least compression and requires less execution time. 9 provides the most compression but requires the most execution time.

- `cParms` - Compression parameter. There is only one parameter for all the compression methods described above.

**Throws:**
- `CDFException` - if a problem occurred in setting the compression type and parameters

---

**getCompression**

```java
public java.lang.String getCompression()
```

Gets the string representation of the compression type and parameters defined for this CDF.

**Returns:**
the string representation of the compression type and parameters (e.g. GZIP.9, RLE.0, etc.) defined for this CDF
Throws:

CDFException - if a problem occurred in getting the compression type and parameters set for this CDF

confirmzMode

public long confirmzMode() throws CDFException

Gets the zMode set for this CDF.

Returns:

'zMODEon2' is always returned since it is the only mode supported by the CDF Java APIs.

Throws:

CDFException - if a problem occurred in getting the zmode set for this CDF file

selectCompressCacheSize

public void selectCompressCacheSize(long compressCacheSize) throws CDFException

Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF). The Concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:

compressCacheSize - the number of 512-byte cache buffers to be used

Throws:

CDFException - if a problem occurs in setting the cache size

confirmCompressCacheSize

public long confirmCompressCacheSize() throws CDFException

Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).

Returns:

the number of 512-byte cache buffers being used

Throws:
selectStageCacheSize

public void selectStageCacheSize(long stageCacheSize)
    throws CDFException

Sets the number of 512-byte cache buffers to be used for the staging scratch file (for the current CDF). The Concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:
    stageCacheSize - the Number of cache buffers to be used

Throws:
    CDFException - if a problem occurs in setting the cache size

confirmStageCacheSize

public long confirmStageCacheSize()
    throws CDFException

Gets the number of 512-byte cache buffers defined for the staging scratch file.

Returns:
    the number of 512-byte cache buffers defined for the staging scratch file

Throws:
    CDFException - if a problem occurs in getting the number of cache buffers defined for the staging scratch file

getName

public java.lang.String getName()

Gets the name of this CDF.

Specified by:
    getName in interface CDFObject

Returns:
    the name of this CDF
rename

public void rename(java.lang.String path)

Renames the current CDF. It's here because CDF.java implements the CDFObject interface that defines three methods: rename, delete, getname. This method doesn't do anything now, but it will be refined to rename a single-CDF and multi-CDF files in the future.

Specified by:
rename in interface CDFObject

Parameters:
path - the new CDF name to be renamed to

delete

public void delete()
throws CDFException

Deletes this CDF file.

Specified by:
delete in interface CDFObject

Throws:
CDFException - if a problem occurs in deleting this CDF file

save

public void save()
throws CDFException

Saves this CDF file without closing. There are times the users will have to save the contents of a CDF file before some operations can be performed. For example, a CDF file must be saved first before records can be deleted properly for variables that are defined to have sparse and/or compressed records.

Throws:
CDFException - if there was a problem saving the contents of this CDF file

setFileBackward

public static void setFileBackward(long flag)
throws CDFException
Sets the file backward flag so that when a new CDF file is created, it will be created in either in the older V2.7 version or the current library version, i.e., V3.*. It only works for V3,* library. Setting this flag will overwrite environment variable CDF_FILEBACKWARD (or CDF$FILEBACKWARD on OpenVMS) if it is set. All CDF files created after this static method call will be affected.

**Parameters:**

- **flag** - The flag indicates whether to create a new CDF(s) in the backward version. BACKWARDFILEon means a backward file(s) is to be created and BACKWARDFILEoff means a V3.* file(s) is to be created.

**Throws:**

- CDFException - if there was a problem

---

**getFileBackward**

```java
public static boolean getFileBackward()
```

Gets the file backward flag.

**Returns:**

The flag indicating whether the CDF file was created in the older V2.7 version. It is only applicable for V3.* library. Returns true if backward files are to be created, false otherwise.

---

**getFileBackwardEnvVar**

```java
public static int getFileBackwardEnvVar() throws CDFException
```

Gets the value of the CDF_FILEBACKWARD environment variable.

**Returns:**

1 if the environment variable is set to true, 0 if not set or set to anything else.

**Throws:**

- CDFException - if there was a problem

---

**getLeapSecondsTableEnvVar**

```java
public static java.lang.String getLeapSecondsTableEnvVar() throws CDFException
```

Gets the the CDF_LEAPSECONDSTABLE (or CDF$LEAPSECONDSTABLE on VMS) environment variable.
getChecksumEnvVar

public static long getChecksumEnvVar() throws CDFException

Gets the value of the CDF_CHECKSUM environment variable.

Returns:
1 if the environment variable is set to MD5, 0 if not set or set to anything else.

Throws:
CDFException - if there was a problem

setValidate

public static void setValidate(long mode) throws CDFException

Sets the file validation mode so that when a CDF file is open, it will be validated accordingly. Setting this flag will overwrite environment variable CDF_VALIDATE (or CDF$VALIDATE on OpenVMS) if it is set. All CDF files open after this static method call will be applied.

Parameters:
mode - The mode indicates whether to validate CDF(s) while open. VALIDATEFILEon means all files are subjected to validation. VALIDATEFILEoff means no data validation will be performed.

Throws:
CDFException - if there was a problem

getValidate

public static boolean getValidate() gets the file validation mode.

Returns:
The mode indicating whether the CDF file is to be validated when it is open. Returns true if it will be validated, false otherwise.

**getStatus**

public long **getStatus**()

Gets the status of the most recent CDF JNI/library function call. This value can be examined and appropriate action can be taken.

The following example sends a signal to the JNI code to write a single data to the current CDF. JNI in turn performs the requested operation. It then checks to see whether the requested operation was successfully performed or not.

```java
variable.putSingleData(recNum, dimIndicies, data);
long status = cdf.getStatus();
if (status != CDF_OK) {
    String statusText = CDF.getStatusText(status);
    System.out.println ("status = "+statusText);
}
```

**Returns:**

the status of the most recent CDF JNI/library function call

**getStatusText**

public static java.lang.String **getStatusText**(long statusCode)

Gets the status text of the most recent CDF JNI/library function call.

The following example shows how to obtain the text representation of the status code returned from the getStatus method:

```java
long status = cdf.getStatus();
if (status != CDF_OK) {
    String statusText = CDF.getStatusText(status);
    System.out.println ("status = "+statusText);
}
```

**Parameters:**

statusCode - status code to be translated

**Returns:**
she string representation of the passed status code

---

**setInfoWarningOff**

```java
class CDF {
    public void setInfoWarningOff()
    {
        Sets the informational (status code > 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function off. This is the default when a file is opened or created.
    }
}
```

---

**setInfoWarningOn**

```java
class CDF {
    public void setInfoWarningOn()
    {
        Sets the informational (status code > 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function on.
    }
}
```

---

**toString**

```java
class CDF {
    public java.lang.String toString()
    {
        Gets the name of this CDF.
    }
}
```

**Overrides:**
- `toString` in class `java.lang.Object`

**Returns:**
- the name of this CDF

---

**finalize**

```java
class CDF {
    public void finalize()
    {
        throws java.lang.Throwable
    }
}
```

**Do the necessary cleanup when garbage collector reaps it.**

**Overrides:**
- `finalize` in class `java.lang.Object`

**Throws:**
- `java.lang.Throwable` - if there was a problem doing cleanup
getDelegate

public CDFDelegate getDelegate()

This is a placeholder for future expansions/extensions.

**Returns:**
CDFDelegate object

setDelegate

public void setDelegate(CDFDelegate delegate)

This is a placeholder for future expansions/extensions.

getAttributeID

public long getAttributeID(java.lang.String attrName)

Gets the id of the given attribute.

**Parameters:**
attrName - the name of the attribute to check

**Returns:**
the id of the named attribute if it exists, -1 otherwise

getAttribute

public Attribute getAttribute(long attrNum)

throws CDFException

Gets the attribute for the given attribute number.

**Note:** The attrNum may not necessarily correspond to the attribute number stored in the CDF file.

**Parameters:**
attrNum - the attribute number to get

**Returns:**
the Attribute object that corresponds to the requested attribute number

Throws:

CDFException - if the supplied attribute number does not exist

---

**getAttribute**

public Attribute getAttribute(java.lang.String attrName)

throws CDFException

Gets the attribute for the given attribute name.

The following example retrieves the attribute named "ValidMin":

```
Attribute validMin = cdf.getAttribute("ValidMin");
```

**Parameters:**

attrName - the name of the attribute to get

**Returns:**

the Attribute object that corresponds to the requested attribute name

**Throws:**

CDFException - if the supplied attribute name does not exist

---

**getAttributes**

public java.util.Vector getAttributes()

Gets all the global and variable attributes defined for this CDF. The following example retrieves all the global and variable attributes:

```
Vector attr = cdf.getAttributes();
```

**Returns:**

a vector that contains the global and variable attributes defined in this CDF

---

**getGlobalAttributes**

public java.util.Vector getGlobalAttributes()

---
Gets the global attributes defined for this CDF.

**Returns:**
A vector that contains the global attributes defined in this CDF

---

**getVariableAttributes**

```java
public java.util.Vector getVariableAttributes()
```

Gets the variable attributes defined for this CDF.

**Returns:**
A vector that contains the variable attributes defined in this CDF.

---

**getOrphanAttributes**

```java
public java.util.Vector getOrphanAttributes()
```

Gets the variable attributes defined for this CDF that are not associated with any variables.

**Returns:**
A vector that contains the empty variable attributes defined in this CDF.

---

**getVariableID**

```java
public long getVariableID(java.lang.String varName)
```

Gets the ID of the given variable.

**Parameters:**
- `varName` - the name of the variable to check

**Returns:**
-1 if the variable does not exist. The variable id if the variable does exist.

---

**getVariable**

```java
public Variable getVariable(long varNum)
```
getVariable

public Variable getVariable(java.lang.String varName)
    throws CDFException

    Gets the variable object for the given variable name.

    The following example retrieves a variable called "Longitude":

        Variable longitude = cdf.getVariable("Longitude");

    Parameters:
        varName - the variable name to get

    Returns:
        the variable object that corresponds to the variable name

    Throws:
        CDFException - if the supplied variable name does not exist

getVariables

public java.util.Vector getVariables()

    Gets the z variables defined for this CDF.

    Note: Since all CDFs opened or created with the CDFJava APIs are placed into zMODE 2, there are no rVariables. All variables are treated as zVariables.
Returns:
a Vector containing all the z variables defined in this CDF

getNumVars

public long getNumVars()

Gets the number of Z variables defined for this CDF.

Note: Since all CDFs opened or create with the CDFJava APIs are placed into zMODE 2, there are no rVariables. All variables are treated as zVariables.

getRecord

public java.util.Vector getRecord(long recNum,
                                   java.lang.String[] strVars)
throws CDFException

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:
recNum - the record number to retrieve data from
strVars - the variable (array of variable names) to retrieve data from

Returns:
the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:
CDFException - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.
Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

**Parameters:**
- `recNum` - the record number to retrieve data from
- `strVars` - the variable (array of variable names) to retrieve data from
- `status` - the individual status (array of statuses) for reading each variable record

**Returns:**
the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

**Throws:**
- `CDFException` - if there was a problem getting a record

**Note:** A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

The following example reads the 2nd record from Longitude and Temperature and prints their contents.

```java
String[] strVars = {"Longitude", "Temperature"};
Vector record;
long[] status = new long[2];
record = cdf.getRecord(1L, strVars, status);

// Check the contents of the 'status' array - optional

// var: Longitude - data type: CDF_UINT2, dimensionality: 1:[3]
System.out.print("2nd record of Longitude -- ");
for (int i=0; i < 3; i++)
    System.out.print(((int[])record.elementAt(0))[i]+" ");
System.out.println(" ");

// var: Temperature -- data type: CDF_REAL4, dimensionality: 1:[3]
System.out.print("2nd record of Temperature -- ");
for (int i=0; i < 3; i++)
    System.out.print(((float[])record.elementAt(1))[i]+" ");
System.out.println(" ");
```
getRecord

public java.util.Vector getRecord(long recNum,
    long[] varIDs)
    throws CDFException

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:
  recNum - the record number to retrieve data from

  varIDs - the variable IDs (array of variable IDs) to retrieve data from

Returns:
  the requested record in a Java vector that contains the variables' data. The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:
  CDFException - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

getRecord

public java.util.Vector getRecord(long recNum,
    long[] varIDs,
    long[] status)
    throws CDFException

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:
  recNum - the record number to retrieve data from

  varIDs - the variable IDs (array of variable IDs) to retrieve data from

  status - the individual status (array of statuses) for reading each variable record

Returns:
  the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

**Throws:**

`CDFException` - if there was a problem getting a record

**Note:** A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

The following example reads the 2nd record from Longitude (varIds[0]) and Temperature (varIDs[1]) and prints their contents.

```java
long[] varIDs = {2, 10};    // Obtained from Variable.getID()
Vector record;
long[] status = new long[2];
record = cdf.getRecord(1L, varIDs, status);

// Check the contents of the 'status' array - optional
// var: Longitude - data type: CDF_UINT2, dimensionality: 1:[3]
System.out.print ("    2nd record of Longitude -- ");
for (int i=0; i < 3; i++)
    System.out.print (((int[])record.elementAt(0))[i]+" ");
System.out.println ("");

// var: Temperature - data type: CDF_REAL4, dimensionality: 1:[3]
System.out.print ("    2nd record of Temperature -- ");
for (int i=0; i < 3; i++)
    System.out.print (((float[])record.elementAt(1))[i]+" ");
System.out.println ("");
```

---

**putRecord**

```java
public void putRecord (long recNum,
                      java.lang.String[] strVars,
                      java.util.Vector myData)
    throws CDFException
```

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

**Parameters:**

- `recNum` - the record number to write data to
- `strVars` - the variable (array of variable names) to write data to
myData - a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector
corresponds to the second variable's record, and so on.

Throws:

CDFException - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the
operation will not be completed. Nothing will be done if the element counts of parameters don't match.

putRecord

public void putRecord(long recNum,
                       java.lang.String[] strVars,
                       java.util.Vector myData,
                       long[] status)
throws CDFException

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is
a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's
data one at a time.

Parameters:

recNum - the record number to write data to

strVars - the variable (array of variable names) to write data to

myData - a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector
corresponds to the second variable's record, and so on.

status - the individual status (array of statuses) for writing each variable record

Throws:

CDFException - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the
operation will not be completed. Nothing will be done if the element counts of parameters don't match.

The following example writes the contents of a record (which consists of two CDF variables - Longitude and
Temperature) to record number 2.

    String[] strVars = {"Longitude",     // variable names in CDF
                        "Temperature"};

    // Longitude -- data type: CDF_UINT2 dimensionality: 1:[3]
```java
int[] longitude_data = {333, 444, 555};

// Temperature -- data type: CDF_FLOAT dimensionality: 0:
Float temperature_data = new Float((float)999.99);

Vector record = new Vector();
record.add(longitude_data);
record.add(temperature_data);

cdf.putRecord(1L, strVars, record);  // Write a record to record #2
```

**putRecord**

**public void putRecord**(long recNum,  
long[] varIDs,  
java.util.Vector myData)  
throws CDFException

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

**Parameters:**
recNum - the record number to write data to

varIDs - the variable IDs (array of variable IDs) to write data to

myData - a Java vector that contains the variables' data. The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

**Throws:**
CDFException - if there was a problem writing the record for any of the variables

**Note:** Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.
Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

**Parameters:**

- **recNum** - the record number to write data to
- **varIDs** - the variable IDs (array of variable IDs) to write data to
- **myData** - a Java vector that contains the variables' data. The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.
- **status** - the individual status (array of statuses) for writing each variable record

**Throws:**

- **CDFException** - if there was a problem writing the record for any of the variables

**Note:** Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.

The following example writes the contents of a record (which consists of two CDF variables - Longitude and Temperature) by using variable IDs (instead of variable names) to record number 2.

```java
long[] varIDs = {3, 9};  // Can be obtained from variable.getID()
// Longitude -- data type: CDF_UINT2 dimensionality: 1:[3]
int[] longitude_data = {333, 444, 555};

// Temperature -- data type: CDF_FLOAT dimensionality: 0:[]
Float temperature_data = new Float((float)999.99);

Vector record = new Vector();
record.add(longitude_data);
record.add(temperature_data);
cdf.putRecord(1L, varIDs, record);  // Write a record to record #2
```

---

**setChecksum**

```java
public void setChecksum(long checksum)
    throws CDFException
```

Specifies the checksum option applied to the CDF.
getChecksum

public long getChecksum()

Gets the checksum method, if any, applied to the CDF.

Returns:
the checksum method used for this CDF. Currently, it returns NONE_CHECKSUM (0) if no checksum is used; MD5_CHECKSUM (1) if MD5 method is used;

Throws:
CDFException - if there was a problem getting the checksum or other vital information from this CDF file

verifyChecksum

public long verifyChecksum() throws CDFException

Verifies the data integrity of the CDF file from its checksum.

Returns:
The status of data integrity check through its checksum. it should return CDF_OK if the integrity check is fine. Or, it may return a value of CHECKSUM_ERROR indicating the data integrity was compromised. Or, it may return other CDF error if it has problem reading the CDF data file(s). No need to use this method as when the file is open, its data integrity is automatically checked with the used checksum method.

Throws:
CDFException - if there was a problem getting the checksum or other vital information from this CDF file
public class CDFData

extends java.lang.Object
implements CDFObject, CDFConstants

This class acts as the glue between the Java code and the Java Native Interface (JNI) code. This class applies only to the Variable object. It handles its data. This class translates a multi-dimensional array data into a 1-dimensional (1D) array prior to sending data to the JNI code for processing. Similarly, data retrieved in 1D array from the JNI code is properly dimensioned for usage or further manipulation.

Version:
1.0, 2.0 03/18/05 Selection of current CDF and variable are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called., 3.0 06/09/09 The number of dimenions returned from the get method depends on the variable dimensions and dimensional elements: e.g., 2-dim (2x1 or 1x2) will have a 2-dim, not 1-dim, object returned, while 2-dim (1x1) returns an object of a single item, not an array.

See Also:
Variable, CDFException
Method Summary

void **delete**()

    See the description of the getName() method in this class.

void **dump**()

    Dump data information and values, one row at a time, to the stdErr.

void **dumpData**()

    Dumps variable data, one row at a time per record.

java.lang.Object **getData**()

    Returns an object that is properly dimensioned.

long[] **getDimCounts**()

    Gets the value of the dimension counts that represents the number of elements read or write starting at the location for a hyper get/put function.

long[] **getDimIndices**()

    Gets the starting dimension index within a record for a hyper get/put function.
### CDFData

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDimIntervals()</code></td>
<td>Gets the value of the dimension intervals that represent the number of elements to skip between reads or writes for a hyper get/put function.</td>
</tr>
<tr>
<td><code>getDimSizes()</code></td>
<td>Gets the dimension sizes of this variable.</td>
</tr>
<tr>
<td><code>getName()</code></td>
<td>CDFData implements CDFObject to enable CDFDelegate calls.</td>
</tr>
<tr>
<td><code>getnDims()</code></td>
<td>Gets the dimensionality of this variable.</td>
</tr>
<tr>
<td><code>getRawData()</code></td>
<td>Returns an object of a 1-dimensional array, which presents a sequence of raw data values retrieved and presented by JNI from a CDF file.</td>
</tr>
<tr>
<td><code>getRecCount()</code></td>
<td>Gets the number of records to read or write for a hyper get/put function.</td>
</tr>
<tr>
<td><code>getRecInterval()</code></td>
<td>Gets the number of records to skip for a hyper get/put function.</td>
</tr>
<tr>
<td><code>getRecStart()</code></td>
<td>Gets the record number at which a hyper get/put function starts.</td>
</tr>
<tr>
<td><code>rename(java.lang.String name)</code></td>
<td>See the description of the <code>getName()</code> method in this class.</td>
</tr>
</tbody>
</table>

Methods inherited from class `java.lang.Object`:

- `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

### Method Detail

#### `getData`

```java
public java.lang.Object getData()
```

Returns an object that is properly dimensioned. The returned object can be casted in an application for usage or further manipulation.

The following example retrieves the Temperature data. The user should know how the data was stored before casting the generic object to a variable.

```java
Variable var = cdf.getVariable("Temperature");
```
CDFData data = var.getHyperDataObject (recNum, recCount, recInterval, dimIndices, dimSizes, dimCounts);

long[][] temperature = (long [][]) data.getData();

Returns:
a generic Object that is properly dimensioned

getRawData

public java.lang.Object getRawData()

Returns an object of a 1-dimensional array, which presents a sequence of raw data values retrieved and presented by JNI from a CDF file. The data stream may or may not reflect how the data is stored in the file, depending on the file majority. It is up to the calling application to decipher the data stream. Note: for column-major CDF files, the sequence of the returned data stream is reordered so data can be properly assigned into Java's arrays. Normally, getData method should be called so the retrieved data are properly dimensioned.

The following example retrieves the 2-D Temperature data. The user should know how to organize the data, e.g., number of records, row/column major, data type, data value sequence, etc.

Variable var = cdf.getVariable("Temperature");
CDFData data = var.getHyperDataObject (recNum, recCount, recInterval, dimIndices, dimSizes, dimCounts);

long[] temperature = (long []) data.getRawData();

Returns:
a generic Object that is properly dimensioned

getnDims

public int getnDims()
CDFData

Gets the dimensionality of this variable.

```java
Variable var = cdf.getVariable("Temperature");
CDFData data = var.getHyperDataObject (recNum,
    recCount,
    recInterval,
    dimIndicies,
    dimSizes,
    dimCounts);

long[][] temperature = (long [][]) data.getData();
nDims = data.getnDims();   // Gives the dimensionality of temperature
```

**Returns:**
the dimensionality of this variable

---

### getDimSizes

**public int[] getDimSizes()**

Gets the dimension sizes of this variable. For example, 3 X 10 (3 rows and 10 columns) two-dimensional array is returned as an one-dimensional integer array, containing 3 in the first element and 10 in the second element.

**Returns:**
the dimension sizes of this variable

---

### getRecStart

**public long getRecStart()**

Gets the record number at which a hyper get/put function starts.

**Returns:**
the starting record number for a hyper get/put function

---

### getRecCount
public long getRecCount()

Gets the number of records to read or write for a hyper get/put function.

**Returns:**
the number of records involved for a hyper get/put function involves

---

**getRecInterval**

public long getRecInterval()

Gets the number of records to skip for a hyper get/put function. The record interval of 1 represents every record. The value of 2 represents every other record, the value of 3 represents every third record and so on.

**Returns:**
the value of record interval

---

**getDimIndices**

public long[] getDimIndices()

Gets the starting dimension index within a record for a hyper get/put function. Dimension index indicates where the data search started from within a record. Let's say a record is comprised of a 2x5 two-dimensional array (2 rows and 5 columns). If the index returned from this method has a value of {1,0}, then the data search was performed starting at the first element of the second row. Similarly, the value of {0,0} represents that the data search search was performed starting at the first element of the first record.

**Returns:**
the dimension index for this variable

---

**getDimCounts**

public long[] getDimCounts()

Gets the value of the dimension counts that represents the number of elements read or write starting at the location for a hyper get/put function.

**Returns:**
getDimIntervals

public long[] getDimIntervals()

Gets the value of the dimension intervals that represent the number of elements to skip between reads or writes for a hyper get/put function. The value of 1 represents every element. The value of 2 represents every other element, and the value of 3 represents every third element and so on.

Returns:
  the dimension intervals for this variable

dumpData

public void dumpData()

Dumps variable data, one row at a time per record. This is a generic utility for dumping data to a screen. Data can be scalar or 1-dimensional or multi-dimensional array of any data type.

The following example retrieves the first record, comprised of 3x5 (3 rows and 5 columns) array, into a generic object and dumps its contents to screen one row at a time. In this case three rows will be displayed on a screen, each row containing 5 elements.

    CDFData data;
    long[] dimIndices   = {0,0};
    long[] dimIntervals = {3,5};
    long[] dimSizes     = {1,1};

    data = var.getHyperDataObject(0L,         // record start
                                 1,          // record counts
                                 1,          // record interval
                                 dimIndices,  
                                 dimSizes,    
                                 dimIntervals);

    data.dumpData();
dump

public void dump()

Dump data information and values, one row at a time, to the stdErr. This method is provided for debugging purposes only. The information is printed in the following manner: /nDims:[sizes] recStart/recCount/recInterval/dimIndices/dimsSizes/dimIntervals/dataArraySignature

getName

public java.lang.String getName()

CDFData implements CDFObject to enable CDFDelegate calls. CDFObject specifies the following three methods: getName(), rename(String), and delete(). Since CDFData implements CDFObject, it must have the methods defined in CDFObject. That's why this method is here; it doesn't do anything.

Specified by:
   getName in interface CDFObject

Returns:
   the name of the current object

rename

public void rename(java.lang.String name)
   throws CDFException

See the description of the getName() method in this class.

Specified by:
   rename in interface CDFObject

Parameters:
   name - the new object name

Throws:
   CDFException - No exception is thrown since this method is a placeholder

delete

public void delete()
throws CDFException

See the description of the getName() method in this class.

Specified by:
   delete in interface CDFObject

Throws:
   CDFException - No exception is thrown since this method is a placeholder
public class CDFNativeLibrary

extends java.lang.Object

implements CDFDelegate

This class implements the method that act as the gateway between the CDF Java APIs and the CDF library.

Version:
Version 1.0

Constructor Summary

CDFNativeLibrary()

Method Summary

void cdflib(CDF theCDF, CDOBject cdfObject, java.util.Vector cmds)
Calls the Java Native Interface (JNI) program, cdfNativeLibrary.c.
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait

Constructor Detail

CDFNativeLibrary

public CDFNativeLibrary()

Method Detail
cdflib

public void cdflib(CDF theCDF,
CDFObject cdfObject,
java.util.Vector cmds)
throws CDFException

Calls the Java Native Interface (JNI) program, cdfNativeLibrary.c. This method is internal and called by various core CDF Java programs.

End users should never call this method from their applications.

Specified by:
cdflib in interface CDFDelegate
Parameters:
theCDF - the CDF being dealt with
cdfObject - the calling program/object (e.g. Variable.java, Attribute.java, etc.)
cmds - a vector that contains the CDFlib commands to be executed

Throws:
CDFException - if a problem occurs while executing the requested CDFlib commands
in cdfNativeLibrary.c.
public interface CDFDelegate

This class defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library. The CDFNativeLibrary class that implementing this interface will cause the JNI to be loaded. This class is available only to the CDF object that uses the CDFDelegate to make requests to JNI. All CDF's other objects, i.e., Attribute, Entry, Variable (and its CDFData), need to refer to the containing CDF object to make requests.

Version:
1.0

See Also:
CDFNativeLibrary

Method Summary

<table>
<thead>
<tr>
<th>void</th>
<th>cdflib(CDF theCDF, CDFObject cdfObject, java.util.Vector cmds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.</td>
</tr>
</tbody>
</table>

Method Detail
void cdflib(CDF theCDF,  
    CDFObject cdfObject,  
    java.util.Vector cmds)  
    throws CDFException

Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library. This method is responsible for sending Java's request to the CDF library and returning the results from the CDF library to the Java side.

Parameters:
  theCDF - the current CDF to be processed
  cdfObject - the calling CDF object (e.g. Attribute, variable, etc.)
  cmds - a Vector that contains the CDF internal interface library commands to be executed

Throws:
  CDFException - if an error occurs processing the requested commands in JNI
public class CDFTools

extends java.lang.Object

implements CDFConstants

CDFTools.java Created: Tue Nov 24 16:14:50 1998

Version:
$Id: CDFTools.java,v 1.1.1.1 2011/03/18 15:56:55 liu Exp $

---

Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th>ALL_VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>NAMED_VALUES</td>
</tr>
<tr>
<td>static int</td>
<td>NO_REPORTS</td>
</tr>
<tr>
<td>static int</td>
<td>NO_VALUES</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>static int</td>
<td>NRV_VALUES</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_ERRORS</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_INFORMATION</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_WARNINGS</td>
</tr>
<tr>
<td>static int</td>
<td>RV_VALUES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fields inherited from interface gsfc.nssdc.cdf.CDFConstants</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING,</td>
</tr>
<tr>
<td>ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING,</td>
</tr>
<tr>
<td>ALPHAVMSq_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING,</td>
</tr>
<tr>
<td>ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_,</td>
</tr>
<tr>
<td>ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_,</td>
</tr>
<tr>
<td>ATTR_NAME_TRUNC, ATTR_MAXgENTRIES_, ATTR_MAXrENTRIES_,</td>
</tr>
<tr>
<td>ATTR_MAXzENTRIES_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_,</td>
</tr>
<tr>
<td>ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff,</td>
</tr>
<tr>
<td>BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME,</td>
</tr>
<tr>
<td>BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION,</td>
</tr>
<tr>
<td>BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM,</td>
</tr>
<tr>
<td>BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING,</td>
</tr>
<tr>
<td>BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE,</td>
</tr>
<tr>
<td>BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT,</td>
</tr>
<tr>
<td>BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE,</td>
</tr>
<tr>
<td>BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_VARS, BAD_READONLY_MODE,</td>
</tr>
<tr>
<td>BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE,</td>
</tr>
<tr>
<td>BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME,</td>
</tr>
<tr>
<td>BAD_VAR_NUM, BAD_zMODE, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE,</td>
</tr>
<tr>
<td>CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,</td>
</tr>
<tr>
<td>CANNOT_SPARSE_RECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN,</td>
</tr>
<tr>
<td>CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHE_SIZE, CDF_CHAR,</td>
</tr>
<tr>
<td>CDF_CHECKSUM, CDF_CLOSE_ERROR, CDF_COMPRESSION, CDF_COPYRIGHT,</td>
</tr>
<tr>
<td>CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING, CDF_DELETE_ERROR</td>
</tr>
</tbody>
</table>
LIB_RELEASE, LIB_subINCREMENT, LIB_VERSION, MAC_DECODING,
MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT,
NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NGtoPOSfp0off, NGtoPOSfp0on,
NETWORK_DECODING, NETWORK_ENCODING, NeXt_DECODING, NeXt_ENCODING,
NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION,
NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS,
NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSETRECORDS,
NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY,
NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF,
NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL, OPEN,
OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSETRECORDS,
PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED,
PREV_SPARSETRECORDS, PUT, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE,
READONLYoff, READONLYOn, rENTRY, rENTRY_DATA, rENTRY_DATASPEC,
rENTRY_DATATYPE, rENTRY_EXISTENCE, rENTRY_NAME, rENTRY_NUMELEMS,
RLE_COMPRESSION, RLE_OF_ZEROS, ROW_MAJOR, rVAR, rVAR_ALLOCATEBLOCK,
rVAR_ALLOCATEDFROM, rVAR_ALLOCATEDTO, rVAR_ALLOCATERECS,
rVAR_BLOCKINGFACTOR, rVAR_CACHESIZE, rVAR_COMPRESSION, rVAR_DATA,
rVAR_DATASPEC, rVAR_DATATYPE, rVAR_DIMVARYS, rVAR_EXISTENCE,
rVAR_HYPERDATA, rVAR_INITIALRECS, rVAR_MAXallocREC, rVAR_MAXREC,
rVAR_NAME, rVAR_nINDEXENTRIES, rVAR_nINDEXLEVELS,
rVAR_nINDEXRECORDS, rVAR_NUMallocRECS, rVAR_NUMBER,
rVAR_NUMELEMS, rVAR_NUMRECS, rVAR_PADVALUE, rVAR_RECORDS,
rVAR_RECORDVARY, rVAR_RESERVEPERCENT, rVAR_SEQDATA, rVAR_SEQPOS,
rVAR_SPARSEARRAYS, rVAR_SPARSETRECORDS, rVARs_CACHESIZE,
rVARs_DIMCOUNTS, rVARs_DIMINDICES, rVARs_DIMINTERVALS,
rVARs_DIMSIZES, rVARs_MAXREC, rVARs_NUMDIMS, rVARs_RECCOUNT,
rVARs_RECDATA, rVARs_RECINTERVAL, rVARs_RECNUMBER, SAVE,
SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR,
SCRATCH_WRITE_ERROR, SELECT, SGi_DECODING, SGi_ENCODING,
SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED,
STAGE_CACHESIZE, STATUS_TEXT, SUN_DECODING, SUN_ENCODING,
TOO_MANY_PARMS, TOO_MANY_VARS, TT2000_0_STRING_LEN,
TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN,
TT2000_TIME_ERROR, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS,
UNSUPPORTED_OPERATION, VALIDATE, VALIDATEFILEoff, VALIDATEFILEon,
VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR,
VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR,
VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRYEXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCDATATOFROM_, zVAR_ALLOCDATATOREC_, zVAR_ALLOCATERCOUNTS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXALLOCREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_INDEXENTRIES_, zVAR_INDEXLEVELS_, zVAR_INDEXRECORDS_, zVAR_NUMALLOCRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECاويت, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVAR_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

CDFTools()

Method Summary

static void skeletonCDF(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSize)

        skeletonTable produces a skeleton table from a CDF.

static void skeletonTable(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, int cacheSize)

        skeletonTable produces a skeleton table from a CDF.

Methods inherited from class java.lang.Object
Field Detail

NO_VALUES

public static final int NO_VALUES

See Also:

Constant Field Values

NRV_VALUES

public static final int NRV_VALUES

See Also:

Constant Field Values

RV_VALUES

public static final int RV_VALUES

See Also:

Constant Field Values

ALL_VALUES

public static final int ALL_VALUES

See Also:
**NAMED_VALUES**

```java
public static final int NAMED_VALUES
```

See Also:
- Constant Field Values

---

**NO_REPORTS**

```java
public static final int NO_REPORTS
```

See Also:
- Constant Field Values

---

**REPORT_ERRORS**

```java
public static final int REPORT_ERRORS
```

See Also:
- Constant Field Values

---

**REPORT_WARNINGS**

```java
public static final int REPORT_WARNINGS
```

See Also:
- Constant Field Values
public static final int REPORT_INFORMATION

See Also:
   Constant Field Values

Constructor Detail

CDFTools

public CDFTools()

Method Detail

skeletonTable

public static void skeletonTable(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, int cacheSize) throws java.io.IOException, java.lang.InterruptedException

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:
skeletonName - is the pathname of the skeleton table to be created. (Do not enter an
extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named .skt in the current directory.

cdfName - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

**log** - Specifies whether or not messages are displayed as the program executes.

**format** - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable).

**neg2posfp0** - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

**statistics** - Specifies whether or not caching statistics are displayed at the end of each CDF.

**screen** - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

**page** - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

**values** - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

- **CDFTools.NO_VALUES**
  Ignore all NRV data values.
- **CDFTools.NRV_VALUES**
  Put NRV data values in the skeleton table.
- **CDFTools.RV_VALUES**
  Put RV variable values in the skeleton table.
- **CDFTools.ALL_VALUES**
  Put all variable values in the skeleton table.
- **CDFTools.NAMED_VALUES**
  Put named variables values in the skeleton table. This requires that valueList be non-null

**valueList** - the named variables to list values.

**zMode** - Specifies which zMode should be used. May be one of the following...

- 0
Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

reportType - Specifies the types of return status codes from the CDF library which should be reported/displayed. report is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

cacheSize - The number of 512-byte buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. If this qualifier is absent, default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of pairs where is the number of cache buffers and is the type of file. The file 's are as follows: `d' for the dotCDF file, `s' for the staging file, and `c' for the compression scratch file. For example, `200d,100s' specifies 200 cache buffers for the dotCDF file and 100 cache buffers for the staging file. The dotCDF file cache size can also be specified without the `d' for compatibility with older CDF releases (eg. `200,100s'). Note that not all of the file types must be specified. Those not specified will receive a default cache size.

Throws:

java.io.IOException
java.lang.InterruptedException

skeletonCDF

public static void skeletonCDF(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSize) throws java.io.IOException, java.lang.InterruptedException
skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

**Parameters:**

- **skeletonName** - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named .skt in the current directory.

- **cdfName** - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

- **delete** - specifies whether or not the CDF should be deleted if it already exists.

- **log** - Specifies whether or not messages are displayed as the program executes.

- **neg2posfp0** - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

- **statistics** - Specifies whether or not caching statistics are displayed at the end of each CDF.

- **zMode** - Specifies which zMode should be used. May be one of the following...

  0  
  Indicates that zMode is disabled.

  1  
  Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

  2  
  Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden).

- **reportType** - Specifies the types of return status codes from the CDF library which should be reported/displayed. **report** is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

- **cacheSize** - The number of 512-byte buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. If this qualifier is absent, default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of pairs where is the number of cache buffers and is the type of file. The file 's are as follows: `d' for the dotCDF file, `s' for the staging file, and `c' for the compression scratch file. For example, `200d,100s' specifies 200 cache buffers for the dotCDF file and 100
cache buffers for the staging file. The dotCDF file cache size can also be specified without
the `d' for compatibility with older CDF releases (eg. `200,100s'). Note that not all of the
file types must be specified. Those not specified will receive a default cache size.

Throws:

java.io.IOException
java.lang.InterruptedException
public class CDFTT2000

extends java.lang.Object

implements CDFConstants
Constructor Summary
## Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>breakdown(long nanosecSinceJ2000)</code></td>
<td>Breaks a TT2000 epoch value down into its full component parts.</td>
</tr>
<tr>
<td><code>CDFgetLastDateinLeapSecondsTable()</code></td>
<td>This method returns the last date that a leap second was added in the leap second table used in the class.</td>
</tr>
<tr>
<td><code>CDFgetLeapSecondsTable()</code></td>
<td>This method returns the leap seconds table.</td>
</tr>
<tr>
<td><code>CDFgetLeapSecondsTableStatus()</code></td>
<td>This method returns the status code reflecting whether the leap seconds are from a external file, defined by an environment variable, or the leap seconds are based on the hard-coded table in the class.</td>
</tr>
<tr>
<td><code>CDFgetRowsinLeapSecondsTable()</code></td>
<td>This method returns the number of entries in the leap seconds table.</td>
</tr>
<tr>
<td><code>compute(long year, long month, long day, long hour, long minute, long second, long msec, long usec, long nsec)</code></td>
<td>Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.</td>
</tr>
<tr>
<td><code>encode(long nanosecSinceJ2000)</code></td>
<td>Converts an epoch value in TT2000 form into a readable date/time string in ISO 8601 format.</td>
</tr>
<tr>
<td><code>encode(long nanosecSinceJ2000, int format)</code></td>
<td>Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.</td>
</tr>
<tr>
<td><code>fromGregorianTime(java.util.GregorianCalendar gc)</code></td>
<td>This method converts the date/time in a GregorianCalendar class object to TT2000 time.</td>
</tr>
<tr>
<td><code>fromUTCEPOCH(double epoch)</code></td>
<td>Convert an epoch value in CDF_EPOCH to TT2000.</td>
</tr>
<tr>
<td><code>fromUTCEPOCH16(double[] epoch)</code></td>
<td>Convert an epoch data in CDF_EPOCH16 to TT2000.</td>
</tr>
<tr>
<td><code>fromUTCparts(double year, double month, double day)</code></td>
<td>Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.</td>
</tr>
<tr>
<td>Method</td>
<td>Signature</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>static long fromUTCparts</td>
<td>(double year, double month, double day, double hour)</td>
</tr>
<tr>
<td></td>
<td>(double year, double month, double day, double hour, double minute)</td>
</tr>
<tr>
<td></td>
<td>(double year, double month, double day, double hour, double minute, double second)</td>
</tr>
<tr>
<td></td>
<td>(double year, double month, double day, double hour, double minute, double second, double milsec)</td>
</tr>
<tr>
<td></td>
<td>(double year, double month, double day, double hour, double minute, double second, double milsec, double micsec)</td>
</tr>
<tr>
<td></td>
<td>(double year, double month, double day, double hour, double minute, double second, double msec, double usec, double nsec)</td>
</tr>
<tr>
<td>static long fromUTCstring</td>
<td>(java.lang.String string)</td>
</tr>
<tr>
<td>static long parse</td>
<td>(java.lang.String string)</td>
</tr>
<tr>
<td>static java.util.GregorianCalendar toGregorianTime</td>
<td>(long tt2000)</td>
</tr>
<tr>
<td>static double toUTCEPOCH</td>
<td>(long nanoSecSinceJ2000)</td>
</tr>
</tbody>
</table>
### Methods

- **static double toUTCepoch16(long nanoSecSinceJ2000, double[] epoch)**
  
  Convert an epoch in TT2000 value to CDF_EPOCH16 value.

- **static long[] toUTCparts(long nanoSecSinceJ2000)**
  
  Breaks a TT2000 epoch value down into its full component parts.

- **static java.lang.String toUTCstring(long nanoSecSinceJ2000)**
  
  Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

- **static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000)**
  
  Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

- **static java.lang.String toUTCstring(long nanoSecSinceJ2000, int format)**
  
  Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

- **static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000, int format)**
  
  Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

### Constructor Details

**CDFTT2000**

```java
public CDFTT2000()
```

### Method Details

**toUTCparts**

```java
public static long[] toUTCparts(long nanoSecSinceJ2000)
```

Breaks a TT2000 epoch value down into its full component parts.

**Parameters:**

- `nanoSecSinceJ2000` - the epoch value, in nanoseconds since J2000, to break down

**Returns:**
breakdown

public static long[] breakdown(long nanoSecSinceJ2000)

Breaks a TT2000 epoch value down into its full component parts.

Parameters:
  nanoSecSinceJ2000 - the epoch value, in nanoseconds since J2000, to break down

Returns:
  an array of long containing the epoch parts:
  
<table>
<thead>
<tr>
<th>Index</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>year</td>
</tr>
<tr>
<td>1</td>
<td>month</td>
</tr>
<tr>
<td>2</td>
<td>day</td>
</tr>
<tr>
<td>3</td>
<td>hour</td>
</tr>
<tr>
<td>4</td>
<td>minute</td>
</tr>
<tr>
<td>5</td>
<td>second</td>
</tr>
<tr>
<td>6</td>
<td>millisecond</td>
</tr>
<tr>
<td>7</td>
<td>microsecond</td>
</tr>
<tr>
<td>8</td>
<td>nanosecond</td>
</tr>
</tbody>
</table>

fromUTCparts

public static long fromUTCparts(double year,
                               double month,
                               double day)

    throws CDFException
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The day field can contain a fraction of a day.

**Parameters:**
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day in double

**Returns:**
- the TT2000 epoch value in long

**Throws:**
- `CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

---

```java
public static long fromUTCparts(double year,
                                double month,
                                double day,
                                double hour)
throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The hour field can contain a fraction of a hour.

**Parameters:**
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day, a full day in double
- `hour` - the hour in double

**Returns:**
- the TT2000 epoch value in long

**Throws:**
- `CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

---

```java
public static long fromUTCparts(double year,
                                double month,
                                double day,
                                double hour,
                                double minute)
throws CDFException
```

**Parameters:**
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day, a full day in double
- `hour` - the hour in double
- `minute` - the minute in double

**Returns:**
- the TT2000 epoch value in long

**Throws:**
- `CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The minute field can contain a fraction of a minute.

Parameters:
- year - the year, a full year in double
- month - the month, a full month in double
- day - the day, a full day in double
- hour - the hour, a full hour in double
- minute - the minute in double

Returns:
- the TT2000 epoch value in long

Throws:
- CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

```
fromUTCparts
```

public static long fromUTCparts(double year, double month, double day, double hour, double minute, double second) throws CDFException

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The second field can contain a fraction of a second.

Parameters:
- year - the year, a full year in double
- month - the month, a full month in double
- day - the day, a full day in double
- hour - the hour, a full hour in double
- minute - the minute, a full minute in double
- second - the second in double

Returns:
- the TT2000 epoch value in long

Throws:
- CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

```
fromUTCparts
```

public static long fromUTCparts(double year, double month, double day, double hour, double minute, double second) throws CDFException
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The millisecond field can contain a fraction of a millisecond.

Parameters:
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day, a full day in double
- `hour` - the hour, a full hour in double
- `minute` - the minute, a full minute in double
- `second` - the second, a full second in double
- `msec` - the millisecond in double

Returns:
- the TT2000 epoch value in long

Throws:
- `CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

---

**fromUTCparts**

```java
public static long fromUTCparts(double year,
                                double month,
                                double day,
                                double hour,
                                double minute,
                                double second,
                                double milsec,
                                double micsec)
                                  
    throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts. The microsecond field can contain a fraction, which usually indicates the end of the parameter list.

Parameters:
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day, a full day in double
- `hour` - the hour, a full hour in double
- `minute` - the minute, a full minute in double
- `second` - the second, a full second in double
- `msec` - the millisecond, a full millisecond in double
- `micsec` - the microsecond, a full microsecond in double

Throws:
- `CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.
fromUTCparts

```java
public static long fromUTCparts(double year,
                                 double month,
                                 double day,
                                 double hour,
                                 double minute,
                                 double second,
                                 double msec,
                                 double usec,
                                 double nsec)
    throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.

**Parameters:**
- `year` - the year, a full year in double
- `month` - the month, a full month in double
- `day` - the day, a full day in double
- `hour` - the hour, a full hour in double
- `minute` - the minute, a full minute in double
- `second` - the second, a full second in double
- `msec` - the millisecond, a full millisecond in double
- `usec` - the microsecond, a full microsecond in double
- `nsec` - the nanosecond, a full nanosecond in double

**Returns:**
the TT2000 epoch value in long

**Throws:**
CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```java
public static long compute(long year,
                            long month,
                            long day,
                            long hour,
                            long minute,
```

**fromUTCparts**

- `usec` - the microsecond in double

**Returns:**
the TT2000 epoch value in long

**Throws:**
CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.
Computes a TT2000 epoch, nanoseconds since J2000, based on its date/time component parts.

Parameters:
- year - the year, a full year in long
- month - the month, a full month in long
- day - the day, a full day in long
- hour - the hour, a full hour in long
- minute - the minute, a full minute in long
- second - the second, a full second in long
- msec - the millisecond, a full millisecond in long
- usec - the microsecond, a full microsecond in long
- nsec - the nanosecond, a full nanosecond in long

Returns:
- the TT2000 epoch value in long

Throws:
- CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

toUTCEPOCH

public static double toUTCEPOCH(long nanoSecSinceJ2000) throws CDFException

Convert an epoch in TT2000 value to CDF_EPOCH value.

Parameters:
- nanoSecSinceJ2000 - the nanoseconds since J2000

Returns:
- the epoch value

Throws:
- CDFException - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

fromUTCEPOCH

public static long fromUTCEPOCH(double epoch) throws CDFException

Convert an epoch value in CDF_EPOCH to TT2000. Reset the predefined CDF_EPOCH value of -1.0E31, -1.0E-
Parameters:
  epoch - the CDF_EPOCH value
Returns:
  the TT2000 epoch in nanoseconds since J2000
Throws:
  CDFException - an TT2000_TIME_ERROR if date is out of the valid range for TT2000.

toUTCEPOCH16

public static double toUTCEPOCH16(long nanoSecSinceJ2000,
                                   double[] epoch)
                              throws CDFException

Convert an epoch in TT2000 value to CDF_EPOCH16 value.

Parameters:
  nanoSecSinceJ2000 - the nanoseconds since J2000
  epoch - the returned CDF_EPOCH16 value, a double[2] object
Returns:
  the status
Throws:
  CDFException - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

fromUTCEPOCH16

public static long fromUTCEPOCH16(double[] epoch)
                              throws CDFException

Convert an epoch data in CDF_EPOCH16 to TT2000. Reset the predefined CDF_EPOCH value of -1.0E31, -1.0E-31, or the default 0.0 or -0.0 (all are invalid for TT2000) to 0.

Parameters:
  epoch - the CDF_EPOCH16 value, a double[2] object
Returns:
  the TT2000 epoch in nanoseconds since J2000
Throws:
  CDFException - an TT2000_TIME_ERROR if date is out of the valid range for TT2000.
toUTCstring

public static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000)

Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

```
yyyy-mm-ddThh:mm:ss.ccccccccc
```

Examples: 1990-04-01T03:05:02.000000000
           1993-10-10T23:45:49.777888999

Parameters:
    nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in Long object

Returns:
    A string representation of the epoch in ISO 8601

---

toUTCstring

public static java.lang.String toUTCstring(long nanoSecSinceJ2000)

Converts an epoch value in TT2000 form into a readable date/time string of ISO 8601 formats.

```
yyyy-mm-ddThh:mm:ss.ccccccccc
```

Examples: 1990-04-01T03:05:02.000000000
           1993-10-10T23:45:49.777888999

These formats are the same as those expected by fromUTCstring.

Parameters:
    nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in long

Returns:
    A string representation of the epoch

---

toUTCstring

public static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000, int format)

Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

```
Format: 0 dd-mmm-yyyy hh:mm:ss.ccccccccc
```

Parameters:
    nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in Long
    format - the format to be used for the date/time string

Returns:
    A string representation of the epoch in chosen format

---
Examples: 01-Apr-1990 03:05:02.000000000
10-Oct-1993 23:45:49.777888999

Format: 1  yyyymmdd.ccccccccc
Examples: 19900401.1234567890
          19931010.9998887776

Format: 2  yyyymmddhhmmss
Examples: 19900401030502
          19931010234549

Format: 3  yyyy-mm-ddThh:mm:ss.ccccccccc
Examples: 1990-04-01T03:05:02.000000000
          1993-10-10T23:45:49.777888999

This is the default, ISO 8601, output.

These formats are the same as those expected by fromUTCstring.

Parameters:
  nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in Long object
  format - the format (from 0 to 3, as the default), an optional

Returns:
  A string representation of the epoch

toUTCstring

public static java.lang.String toUTCstring(long nanoSecSinceJ2000,
                                           int format)

Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.
This is the default, ISO 8601, output.

These formats are the same as those expected by fromUTCstring.

**Parameters:**
- nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in long format
- format - the format (from 0 to 3, as the default), an optional

**Returns:**
A string representation of the epoch

---

**encode**

**public static java.lang.String encode(long nanoSecSinceJ2000)**

Converts an epoch value in TT2000 form into a readable date/time string in ISO 8601 format.

```
    yyyy-mm-ddThh:mm:ss.ccccccccc
```

Examples:
- 1990-04-01T03:05:02.000000000
- 1993-10-10T23:45:49.777888999

**Parameters:**
- nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in long

**Returns:**
A string representation of the epoch

---

**encode**

**public static java.lang.String encode(long nanoSecSinceJ2000, int format)**

Converts an epoch value in TT2000 form into a readable date/time string of chosen formats.

**Format: 0**
```
    dd-mmm-yyyy hh:mm:ss.ccccccccc
```

Examples:
- 01-Apr-1990 03:05:02.000000000
- 10-Oct-1993 23:45:49.777888999

**Format: 1**
```
    yyyymmdd.ccccccccc
```

Examples:
- 19900401.1234567890
- 19931010.9998887776

**Format: 2**
```
    yyyymmddhhmmss
```

Examples:
- 19900401030502
19931010234549

Format: 3  yyyy-mm-ddThh:mm:ss.ccccccccc
Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
This is the default, ISO 8601, output.

These formats are the same as those expected by fromUTCstring.

Parameters:
- nanoSecSinceJ2000 - the TT2000 epoch value, nanoseconds since J2000, in long format
- the format (from 0 to 3, as the default), an optional

Returns:
- A string representation of the epoch

fromUTCstring

public static long fromUTCstring(java.lang.String string)
    throws CDFException

This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the formats as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

Format: 0  dd-mmm-yyyy hh:mm:ss.ccccccccccc
Examples:  01-Apr-1990 03:05:02.000000000
             10-Oct-1993 23:45:49.777888999

Format: 1  yyyyMMdd.cccccccccccc
Examples:  19900401.1234567890
             19931010.9998887776

Format: 2  yyyyMMddhhmmss
Examples:  19900401030502
             19931010234549

Format: 3  yyyy-mm-ddThh:mm:ss.ccccccccccc
Examples:  1990-04-01T03:05:02.000000000
             1993-10-10T23:45:49.777888999

These formats are the same as those created by toUTCstring.

Parameters:
- string - the epoch in string representation

Returns:
A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

---

**parse**

public static long parse(java.lang.String string) throws CDFException

This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the formats as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

**Format: 0**

dd-mmm-yyyy hh:mm:ss.ccccccccc

Examples: 01-Apr-1990 03:05:02.000000000
10-Oct-1993 23:45:49.777888999

**Format: 1**

yyyymmdd.ccccccccc

Examples: 19900401.1234567890
19931010.9998887776

**Format: 2**

yyyymmddhhmmss

Examples: 19900401030502
19931010234549

**Format: 3**

yyyy-mm-ddThh:mm:ss.ccccccccc

Examples: 1990-04-01T03:05:02.000000000
1993-10-10T23:45:49.777888999

These formats are the same as those created by toUTCstring.

Parameters:

string - the epoch in string representation

Returns:

A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

---

**CDFgetLastDateinLeapSecondsTable**

public static long[] CDFgetLastDateinLeapSecondsTable()

This method returns the last date that a leap second was added in the leap second table used in the class. This can
be used to check whether the table is up-to-date.

**Returns:**
the date (year, month, day) in a long array

### fromGregorianTime

**public static long fromGregorianTime(java.util.GregorianCalendar gc)**

This method converts the date/time in a GregorianCalendar class object to TT2000 time.

**Returns:**
the TT2000 time

**Throws:**
CDFException - a TT2000_TIME_ERROR if a GregorianCalendar component value is detected.

### toGregorianTime

**public static java.util.GregorianCalendar toGregorianTime(long tt2000)**

This method converts the date/time in TT2000 to a GregorianCalendar class object.

**Returns:**
a GregorianCalendar object

### CDFgetLeapSecondsTableStatus

**public static int CDFgetLeapSecondsTableStatus()**

This method returns the status code reflecting whether the leap seconds are from a external file, defined by an environment variable, or the leap seconds are based on the hard-coded table in the class.

**Returns:**
status 1 if form a file, 0 hard-coded
public static double[][] CDFgetLeapSecondsTable()

This method returns the leap seconds table.

**Returns:**

table The table contents of the leap seconds

---

**CDFgetRowsinLeapSecondsTable**

public static int CDFgetRowsinLeapSecondsTable()

This method returns the number of entries in the leap seconds table.

**Returns:**

entryCnt The entry count in the leap seconds table
public class CDFUtils

extends java.lang.Object
implements CDFConstants

This class contains the handy utility routines (methods) called by the core CDF Java APIs.

Version:
1.0

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants
Constructor Summary

CDFUtils ()

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static boolean cdfFileExists(java.lang.String fileName)</td>
<td>Checks the existence of the given CDF file name.</td>
</tr>
<tr>
<td>static long getDataTypeValue(java.lang.String cdfDataType)</td>
<td>Gets the long value of the given CDF data type in string.</td>
</tr>
<tr>
<td>static long getLongChecksum(java.lang.String checksum)</td>
<td>Gets the long value of the given CDF's checksum in string.</td>
</tr>
<tr>
<td>static long getLongCompressionType(java.lang.String compressionType)</td>
<td>Gets the long representation of the given CDF compression type in string.</td>
</tr>
<tr>
<td>static long getLongEncoding(java.lang.String encodingType)</td>
<td>Gets the long value of the given CDF encoding type in string.</td>
</tr>
<tr>
<td>static long getFormat(java.lang.String formatType)</td>
<td>Gets the long value of the given CDF file format in string.</td>
</tr>
<tr>
<td>Method</td>
<td>Parameters</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getLongMajority</code></td>
<td><code>java.lang.String majorityType</code></td>
</tr>
<tr>
<td><code>getLongSparseRecord</code></td>
<td><code>java.lang.String sparseRecordType</code></td>
</tr>
<tr>
<td><code>getNumElements</code></td>
<td><code>long dataType, java.lang.Object data</code></td>
</tr>
<tr>
<td><code>getSignature</code></td>
<td><code>java.lang.Object obj</code></td>
</tr>
<tr>
<td><code>getStringChecksum</code></td>
<td><code>CDF cdf</code></td>
</tr>
<tr>
<td><code>getStringChecksum</code></td>
<td><code>long checksumType</code></td>
</tr>
<tr>
<td><code>getStringCompressionType</code></td>
<td><code>CDF cdf</code></td>
</tr>
<tr>
<td><code>getStringCompressionType</code></td>
<td><code>long compressionType</code></td>
</tr>
<tr>
<td><code>getStringCompressionType</code></td>
<td><code>Variable var</code></td>
</tr>
<tr>
<td><code>getStringData</code></td>
<td><code>java.lang.Object data</code></td>
</tr>
<tr>
<td><code>getStringData</code></td>
<td><code>java.lang.Object data, int epochType</code></td>
</tr>
<tr>
<td><code>getStringData</code></td>
<td><code>java.lang.Object data, java.lang.String separator</code></td>
</tr>
<tr>
<td><code>getStringData</code></td>
<td><code>java.lang.Object data, java.lang.String separator, int epochType</code></td>
</tr>
<tr>
<td><code>getStringDataType</code></td>
<td><code>Entry entry</code></td>
</tr>
<tr>
<td><code>getStringDataType</code></td>
<td><code>long cdfDataType</code></td>
</tr>
<tr>
<td><code>getStringDataType</code></td>
<td><code>Variable var</code></td>
</tr>
<tr>
<td><code>getStringDecoding</code></td>
<td><code>CDF cdf</code></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringDecoding</code> (long decodingType)</td>
<td>Gets the string value of the given CDF decoding type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringEncoding</code> (CDF cdf)</td>
<td>Get the string value of the given CDF's encoding type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringEncoding</code> (long encodingType)</td>
<td>Gets the string value of the given CDF encoding type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringFormat</code> (CDF cdf)</td>
<td>Gets the string value of the given CDF's file format.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringFormat</code> (long formatType)</td>
<td>Gets the string value of the given CDF's file format.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringMajority</code> (CDF cdf)</td>
<td>Gets the string value of the given CDF file's majority.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringMajority</code> (long majorityType)</td>
<td>Gets the string value of the given CDF majority.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringSparseRecord</code> (long sparseRecordType)</td>
<td>Gets the string value of the given sparse record type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringSparseRecord</code> (Variable var)</td>
<td>Gets the string value of the given variable's sparse record type.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data)</td>
<td>Prints the value of the given data on the screen.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data, int which)</td>
<td>Prints the value of the given data on the screen.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data, int which, boolean iso8601)</td>
<td>Prints the value of the given data on the screen.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data, java.io.PrintWriter outWriter)</td>
<td>Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data, java.io.PrintWriter outWriter, int which)</td>
<td>Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.</td>
</tr>
<tr>
<td>static void <code>printData</code> (java.lang.Object data, java.io.PrintWriter outWriter, int which, boolean iso8601)</td>
<td>Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.</td>
</tr>
</tbody>
</table>
CDFUtils

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CDFUtils

public CDFUtils()  

Method Detail

getSignature

public static java.lang.String getSignature(java.lang.Object obj)  

Gets the java signature of the given object.

NOTE: Java primitive data types (e.g. int, long, byte, etc.) are not Objects. Thus they must be passed-in as an Object by using a wrapper (e.g. Integer(23)).

<table>
<thead>
<tr>
<th>Signature</th>
<th>Java Programming Language Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Z</td>
<td>array of boolean</td>
</tr>
<tr>
<td>[B</td>
<td>array of byte</td>
</tr>
<tr>
<td>[C</td>
<td>array of char</td>
</tr>
<tr>
<td>[S</td>
<td>array of short</td>
</tr>
<tr>
<td>[I</td>
<td>array of int</td>
</tr>
<tr>
<td>[J</td>
<td>array of long</td>
</tr>
<tr>
<td>[F</td>
<td>array of float</td>
</tr>
<tr>
<td>[D</td>
<td>array of double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L fully-qualified-class</th>
<th>fully-qualified class</th>
</tr>
</thead>
<tbody>
<tr>
<td>L fully-qualified-class;</td>
<td>array of fully-qualified class</td>
</tr>
<tr>
<td>java.lang.Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>Ljava.lang.Boolean;</td>
<td>array of Boolean</td>
</tr>
<tr>
<td>java.lang.Byte</td>
<td>Byte</td>
</tr>
<tr>
<td>Ljava.lang.Byte;</td>
<td>array of Byte</td>
</tr>
<tr>
<td>java.lang.Short</td>
<td>Short</td>
</tr>
<tr>
<td>Ljava.lang.Short;</td>
<td>array of Short</td>
</tr>
</tbody>
</table>
java.lang.Integer         Integer
Ljava.lang.Integer;       array of Integer
java.lang.Long            Long
Ljava.lang.Long;          array of Long
java.lang.Float           Float
Ljava.lang.Float;         array of Float
java.lang.Double          Double
Ljava.lang.Double;        array of Double
java.lang.String          String
Ljava.lang.String;        array of String

Parameters:
    obj - the object from which Java signature is retrieved

Returns:
    Java signature of the given object

getNumElements

public static long getNumElements(long dataType,
                      java.lang.Object data)
        throws CDFException

Gets the number of elements contained in the given data object.

Parameters:
    dataType - the CDF data type of the object to be examined
    data - the data object to be examined

Returns:
    If the data is a string: number of characters in the string
    If the data is an array: number of elements in the array
    Otherwise: 1

Throws:
    CDFException - if a problem occurs getting the number of elements

printData
public static void printData(java.lang.Object data)

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
   data - the data to be printed

printData

public static void printData(java.lang.Object data, int which, boolean iso8601)

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
   data - the data to be printed
   which - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000
   iso8601 - the ISO 8601 indicator for EPOCH data

printData

public static void printData(java.lang.Object data, int which)

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
   data - the data to be printed
   which - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

printData

public static void printData(java.lang.Object data,
java.io.PrintWriter outWriter)

Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

The following example will send the contents of the given data to "myoutput.dat".

```
OutputStreamWriter outWriter = null;
PrintWriter out = null;
try {
    outWriter = new OutputStreamWriter("myoutput.dat", "UTF-8");
    out = new PrintWriter(outWriter, true);
} catch (Exception e) {
    System.out.println ("Exception occurred: "+e);
}
CDFUtils.printData (data, out);
```

Parameters:

data - the data to be printed

outWriter - the print writer to which formatted representations of the object/data is printed as a text-output stream

---

printData

public static void printData(java.lang.Object data, java.io.PrintWriter outWriter, int which)

Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

data - the data to be printed

outWriter - the print writer to which formatted representations of the object/data is printed as a text-output stream

which - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000
printData

public static void printData(java.lang.Object data,
             java.io.PrintWriter outWriter,
             int which,
             boolean iso8601)

Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
  data - the data to be printed

  outWriter - the print writer to which formatted representations of the object/data is printed as a text-output stream

  which - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

  iso8601 - the ISO 8601 indicator for EPOCH data

getStringData

public static java.lang.String getStringData(java.lang.Object data)

Returns the string value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
  data - the data to be parsed

Returns:
  The string value of the given data/object.
  If the data is an array, its elements are delimited by a space.

getStringData

public static java.lang.String getStringData(java.lang.Object data,
                                             int epochType)

Returns the string value of the given data. Data can be a java primitive data type, Java Object (non-array), or
1-dimensional array of primitive Java data type.

Parameters:
- data - the data to be parsed
- epochType - epoch type indicator (==1 CDF_EPOCH, ==2 CDF_EPOCH16, ==0 others)

Returns:
The string value of the given data/object.
If the data is an array, its elements are delimited by a space.

getStringData

public static java.lang.String getStringData(java.lang.Object data,
java.lang.String separator)

returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
- data - the data to be parsed
- separator - the delimiter for array elements

Returns:
The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

getStringData

public static java.lang.String getStringData(java.lang.Object data,
java.lang.String separator,
int epochType)

returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:
- data - the data to be parsed
- separator - the delimiter for array elements
**epochType** - Epoch or Epoch16 data type indicator

== 1 for EPOCH, == 2 for EPOCH16, == 3 for TT2000, == 0 for other data types

**Returns:**

The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

---

**getStringDataType**

public static java.lang.String **getStringDataType**(Variable var)

Gets the string value of the CDF data type for the given variable.

**Parameters:**

var - the CDF variable to be examined

**Returns:**

See getStringDataType (long cdfDataType) for possible return values.

---

**getStringDataType**

public static java.lang.String **getStringDataType**(Entry entry)

Gets the string value of the CDF data type for the given entry.

**Parameters:**

entry - the entry to be examined

**Returns:**

String representation of the entry's CDF data type. See getStringDataType (long cdfDataType) for possible return values.

---

**getStringDataType**

public static java.lang.String **getStringDataType**(long cdfDataType)

Gets the string representation of the given CDF data type.
**Parameters:**

cdfDataType - the CDF data type to be examined and translated

It should be one of the following:

- CDF_BYTE
- CDF_CHAR
- CDF_UCHAR
- CDF_INT1
- CDF_UINT1
- CDF_INT2
- CDF_UINT2
- CDF_INT4
- CDF_UINT8
- CDF_INT8
- CDF_REAL4
- CDF_FLOAT
- CDF_REAL8
- CDF_DOUBLE
- CDF_EPOCH
- CDF_EPOCH16
- CDF_TIME_TT2000

**Returns:**

String representation of cdfDataType. The returned value is one of the valid values describe above for cdfDataType. "UNKNOWN" is returned if invalid cdfDataType is given.

---

**getDataTypeValue**

```java
public static long getDataTypeValue(java.lang.String cdfDataType)
```

Gets the long value of the given CDF data type in string. This is a reverse function from getStringDataType.

**Parameters:**

cdfDataType - the string CDF data type to be examined and translated. It should be one of the following values:

- CDF_BYTE
- CDF_CHAR
- CDF_UCHAR
- CDF_INT1
- CDF_UINT1
- CDF_INT2
- CDF_UINT2
- CDF_INT4
```
>Returns:
long representation of cdfDataType. The returned value is one of the valid values described above for
cdfDataType. -1 is returned if invalid cdfDataType is given.

---

**getStringCompressionType**

```
public static java.lang.String getStringCompressionType(long compressionType)
```

Gets the string representation of the given CDF compression type.

**Parameters:**
compressionType - the CDF compression type to be translated. It should be one of the following:

- NO_COMPRESSION
- RLE_COMPRESSION
- HUFF_COMPRESSION
- AHUFF_COMPRESSION
- GZIP_COMPRESSION

**Returns:**
String representation of compressionType. The returned value is one of the following:

- NONE
- RLE
- Huffman
- Adaptive Huffman
- GZIP
- UNKNOWN (for unknown compressionType)

---

**getLongCompressionType**

```
public static long getLongCompressionType(java.lang.String compressionType)
```

---

Gets the long representation of the given CDF compression type in string.

**Parameters:**

compressionType - the CDF compression type to be translated. It should be one of the following:

- NONE
- RLE
- Huffman
- Adaptive Huffman
- GZIP

**Returns:**

long representation of compressionType. The returned value is one of the following:

- NO_COMPRESSION
- RLE_COMPRESSION
- HUFF_COMPRESSION
- AHUFF_COMPRESSION
- GZIP_COMPRESSION
- -1 (for unknown compressionType)

---

**getStringCompressionType**

```java
public static java.lang.String getStringCompressionType(Variable var)
```

Gets the string representation of the given variable's compression type.

**Parameters:**

var - the variable to be examined

**Returns:**

string representation of the given variable's compression type. See `getStringCompressionType(long compressionType)` for possible return values.

---

**getStringCompressionType**

```java
public static java.lang.String getStringCompressionType(CDF cdf)
```

Gets the string representation of the given CDF file's compression type.

**Parameters:**

cdf - the CDF to be examined
**Returns:**
string representation of the given CDF file's compression type. See getStringCompressionType(long compressionType) for possible return values.

---

### getStringEncoding

```java
public static java.lang.String getStringEncoding(long encodingType)
```

Gets the string value of the given CDF encoding type.

**Parameters:**
- `encodingType` - the CDF encoding type to be examined. It should be one of the following:
  - NETWORK_ENCODING
  - SUN_ENCODING
  - DECSTATION_ENCODING
  - SGi_ENCODING
  - IBMPC_ENCODING
  - IBMRS_ENCODING
  - HOST_ENCODING
  - PPC_ENCODING
  - HP_ENCODING
  - NeXT_ENCODING
  - ALPHAOSF1_ENCODING
  - ALPHAVMSd_ENCODING
  - ALPHAVMSg_ENCODING
  - ALPHAVMSi_ENCODING

**Returns:**
string representation of `encodingType`. The returned value is one of the following:
- NETWORK
- SUN
- DECSTATION
- SGi
- IBMPC
- IBMRS
- HOST
- PPC
- HP
- NeXT
- ALPHAOSF1
- ALPHAVMSd
- ALPHAVMSg
- ALPHAVMSi
- UNKNOWN (for unknown `encodingType`)
getLongEncoding

public static long getLongEncoding(java.lang.String encodingType)

Gets the long value of the given CDF encoding type in string.

Parameters:

- encodingType - the CDF encoding type to be examined. It should be one of the following:
  - NETWORK
  - SUN
  - DECSTATION
  - SGi
  - IBMPC
  - IBMRS
  - HOST
  - PPC
  - HP
  - NeXT
  - ALPHAOOSF1
  - ALPHAVMSd
  - ALPHAVMSg
  - ALPHAVMSi

Returns:

- long representation of encodingType. The returned value is one of the following:
  - NETWORK_ENCODING
  - SUN_ENCODING
  - DECSTATION_ENCODING
  - SGi_ENCODING
  - IBMPC_ENCODING
  - IBMRS_ENCODING
  - HOST_ENCODING
  - PPC_ENCODING
  - HP_ENCODING
  - NeXT_ENCODING
  - ALPHAOOSF1_ENCODING
  - ALPHAVMSd_ENCODING
  - ALPHAVMSg_ENCODING
  - ALPHAVMSi_ENCODING
  - -1 (for unknown encodingType)

getStringEncoding
public static java.lang.String getStringEncoding(CDF cdf)

Get the string value of the given CDF's encoding type.

**Parameters:**
- cdf - the CDF to be examined

**Returns:**
- string representation of the given CDF's encoding type. See getStringEncoding(long encodingType) for possible return values.

---

**getStringDecoding**

public static java.lang.String getStringDecoding(long decodingType) throws CDFException

Gets the string value of the given CDF decoding type.

**Parameters:**
- decodingType - the CDF decoding type to be examined. It should be one of the following:
  - NETWORK_DECODING
  - SUN_DECODING
  - DECSTATION_DECODING
  - SGi_DECODING
  - IBMPC_DECODING
  - IBMRS_DECODING
  - HOST_DECODING
  - PPC_DECODING
  - HP_DECODING
  - NeXT_DECODING
  - ALPHAOSSF1_DECODING
  - ALPHAVMSd_DECODING
  - ALPHAVMSg_DECODING
  - ALPHAVMSi_DECODING
  - -1 (for unknown encodingType)

**Returns:**
- string representation of decodingType. See getStringEncoding (long encodingType) for possible return values.

**Throws:**
getStringDecoding

public static java.lang.String getStringDecoding(CDF cdf) throws CDFException

Gets the string value of the given CDF file's decoding type.

Parameters:

cdf - the CDF to be examined

Returns:

string representation of the given CDF file's decoding type. See getStringEncoding (long encodingType) for possible return values.

Throws:

CDFException - if a problem occurs getting the value of the decoding type defined for the given CDF

getStringMajority

public static java.lang.String(getStringMajority)(long majorityType)

Gets the string value of the given CDF majority.

Parameters:

majorityType - the CDF majority to be translated

Returns:

string representation of majorityType. The returned value is one of the following:

- ROW
- COLUMN
- UNKNOWN (for unknown majorityType)

getLongMajority

public static long getLongMajority(java.lang.String majorityType)


Gets the long value of the given CDF majority.

**Parameters:**
majorityType - the CDF majority to be translated. It should be either ROW or COLUMN

**Returns:**
long representation of majorityType. The returned value is one of the following:
- ROW_MAJOR
- COLUMN_MAJOR
- -1 (for unknown majorityType)

---

**getStringMajority**

public static java.lang.String getStringMajority(CDF cdf)

Gets the string value of the given CDF file's majority.

**Parameters:**
cdf - the CDF to be examined

**Returns:**
string representation of the given CDF file's majority. The returned value is one of the following:
- ROW
- COLUMN

---

**getStringFormat**

public static java.lang.String getStringFormat(long formatType)

Gets the string value of the given CDF's file format.

**Parameters:**
formatType - the CDF file format to be translated. It should be either SINGLE or MULTI

**Returns:**
string representation of formatType. The returned value is either SINGLE, MULTI, or UNKNOWN.

---

**getLongFormat**
public static long `getLongFormat` (java.lang.String formatType)

Gets the long value of the given CDF file format in string.

**Parameters:**
formatType - the CDF file format to be translated. It should be either SINGLE or MULTI.

**Returns:**
long representation of formatType. The returned value is one of the following:
- SINGLE_FILE
- MULTI_FILE
- -1 (for unknown format type)

---

**getStringFormat**

public static java.lang.String `getStringFormat`(CDF cdf)

Gets the string value of the given CDF's file format.

**Parameters:**
cdf - the CDF to be examined

**Returns:**
string representation of given CDF's file format. The returned value is either SINGLE, MULTI, or UNKNOWN.

---

**getStringSparseRecord**

public static java.lang.String `getStringSparseRecord`(long sparseRecordType)

Gets the string value of the given sparse record type.

**Parameters:**
sparseRecordType - the sparse record type to be translated. It should be one of the following:
- NO_SPARSERECORDS
- PAD_SPARSERECORDS
- PREV_SPARSERECORDS

**Returns:**
string representation of sparseRecordType. The returned value is one of the following:
- None
getLongChecksum

public static long getLongChecksum(java.lang.String checksum)

Gets the long value of the given CDF's checksum in string.

Parameters:
- checksum - the checksum string of which to be translated.

Returns:
- long value of checksum type. The returned value is either NONE_CHECKSUM, MD5_CHECKSUM, or OTHER_CHECKSUM.

getStringChecksum

public static java.lang.String getStringChecksum(CDF cdf)

Gets the string value of the given CDF's checksum.

Parameters:
- cdf - the CDF with which its checksum to be translated.

Returns:
- string representation of checksum type. The returned value is either NONE, MD5, or OTHER.

getStringChecksum

public static java.lang.String getStringChecksum(long checksumType)

Gets the string value of the given CDF's checksum.

Parameters:
- checksumType - the CDF checksum to be translated. It should be either NO_CHECKSUM (or NONE_CHECKSUM) or MD5_CHECKSUM

Returns:
- string representation of checksumType. The returned value is either NONE, MD5, or OTHER.
**getLongSparseRecord**

```java
public static long getLongSparseRecord(java.lang.String sparseRecordType)
```

Gets the long value of the given sparse record type in string.

**Parameters:**
- `sparseRecordType` - the sparse record type to be translated. It should be one of the following:
  - None
  - PAD or sRecords.PAD
  - PREV or sRecords.PREV

**Returns:**
- long representation of `sparseRecordType`. The returned value is one of the following:
  - NO_SPARSERECORDS
  - PAD_SPARSERECORDS
  - PREV_SPARSERECORDS
  - -1 (for unknown sparse record type)

---

**getStringSparseRecord**

```java
public static java.lang.String getStringSparseRecord(Variable var)
```

Gets the string value of the given variable's sparse record type.

**Parameters:**
- `var` - the variable to be examined

**Returns:**
- string representation of the given variable's sparse record type. The returned value is one of the following:
  - None
  - PAD
  - PREV
  - UNKNOWN

---

**cdfFileExists**
public static boolean cdfFileExists(java.lang.String fileName)

Checks the existence of the given CDF file name. If the file name doesn't have ".cdf" file extension, it adds ".cdf" suffix at the end of the file name before checking the existence of the file. If the file exists in the current directory, it returns TRUE. Otherwise, FALSE is returned.

Parameters:

fileName - the name of the CDF file to be checked for existence

Returns:

true - if fileName exists in the current directory
false - if fileName doesn't exist in the current directory
This class describes a CDF global or variable attribute entry.

**Note:** In the Java CDF API there is no concept of an rEntry since r variables are not supported. Only z variables are supported since it is far superior and efficient than r variables.

**Version:**
1.0, 2.0 03/18/05 Selection of current CDF, attribute and entry are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called.

**See Also:**
`Attribute`
<table>
<thead>
<tr>
<th>AHUFF_COMPRESSION</th>
<th>ALPHAOSF1_DECODING</th>
<th>ALPHAOSF1_ENCODING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAVMSd_DECODING</td>
<td>ALPHAVMSd_ENCODING</td>
<td>ALPHAVMSg_DECODING</td>
</tr>
<tr>
<td>ALPHAVMSg_ENCODING</td>
<td>ALPHAVMSi_DECODING</td>
<td>ALPHAVMSi_ENCODING</td>
</tr>
<tr>
<td>ATTR_EXISTENCE</td>
<td>ATTR_EXISTS</td>
<td>ATTR_MAXgENTRY</td>
</tr>
<tr>
<td>ATTR_MAXrENTRY</td>
<td>ATTR_NAME</td>
<td>ATTR_NAME_TRUNC</td>
</tr>
<tr>
<td>ATTR_numgENTRIES</td>
<td>ATTR_numrENTRIES</td>
<td>ATTR_numzENTRIES</td>
</tr>
<tr>
<td>ATTR_SCOPE</td>
<td>BACKWARD</td>
<td>BACKWARDFILEoff</td>
</tr>
<tr>
<td>BACKWARDFILEon</td>
<td>BAD_ALLOCATE_RECS</td>
<td>BAD_ARGUMENT</td>
</tr>
<tr>
<td>BAD_Attr_NAME</td>
<td>BAD_Attr_NUM</td>
<td>BAD_BLOCKING_FACTOR</td>
</tr>
<tr>
<td>BAD_CACHE_SIZE</td>
<td>BAD_CODEC_EXTENSION</td>
<td>BAD_CDF_ID</td>
</tr>
<tr>
<td>BAD_CDF_NAME</td>
<td>BAD_CDFSTATUS</td>
<td>BAD_CHECKSUM</td>
</tr>
<tr>
<td>BAD_COMPRESSION_PARM</td>
<td>BAD_DATA_TYPE</td>
<td>BAD_DECODING</td>
</tr>
<tr>
<td>BAD_DIM_COUNT</td>
<td>BAD_DIM_INDEX</td>
<td>BAD_ENCODING</td>
</tr>
<tr>
<td>BAD_ENCODING</td>
<td>BAD_ENTRY_NUM</td>
<td>BAD_FNC_OR_ITEM</td>
</tr>
<tr>
<td>BAD_FORMAT</td>
<td>BAD_INITIAL_RECS</td>
<td>BAD_MAJORITY</td>
</tr>
<tr>
<td>BAD_MALLOC</td>
<td>BAD_NEGtoPOSfp0_MODE</td>
<td>BAD_NUM_DIMS</td>
</tr>
<tr>
<td>BAD_NUM_ELEMS</td>
<td>BAD_NUM_VARS</td>
<td>BAD_READONLY_MODE</td>
</tr>
<tr>
<td>BAD_rec_COUNT</td>
<td>BAD_REC_INTERVAL</td>
<td>BAD_REC_NUM</td>
</tr>
<tr>
<td>BAD_SCRATCH_DIR</td>
<td>BAD_SPARSEARRAYS_PARM</td>
<td>BAD_VAR_NAME</td>
</tr>
<tr>
<td>BAD_VAR_NUM</td>
<td>BAD_ZMODE</td>
<td>CANNOT_ALLOCATE_RECORDS</td>
</tr>
<tr>
<td>CANNOT_CHANGE</td>
<td>CANNOT_COMPRESS</td>
<td>CANNOT_COPY</td>
</tr>
<tr>
<td>CANNOT_COPY</td>
<td>CANNOT_SPARSEARRAYS</td>
<td>CANNOT_SPARSERECORDS</td>
</tr>
<tr>
<td>CDF</td>
<td>CDF_ACCESS</td>
<td>CDF_ATTR_NAME_LEN</td>
</tr>
<tr>
<td>CDF_BYTE</td>
<td>CDF_CACHESIZE</td>
<td>CDF_CHAR</td>
</tr>
<tr>
<td>CDF_CHECKSUM</td>
<td>CDF_CLOSE_ERROR</td>
<td>CDF_COMPRESSION</td>
</tr>
<tr>
<td>CDF_COPYRIGHT</td>
<td>CDF_COPYRIGHT_LEN</td>
<td>CDF_CREATE_ERROR</td>
</tr>
<tr>
<td>CDF_DECODING</td>
<td>CDF_DELETE_ERROR</td>
<td>CDF DOUBLE</td>
</tr>
<tr>
<td>CDF_ENCODING</td>
<td>CDF_EPOCH</td>
<td>CDF_EPOCH16</td>
</tr>
<tr>
<td>CDF_EXISTS</td>
<td>CDF_FLOAT</td>
<td>CDF_FORMAT</td>
</tr>
<tr>
<td>CDF_INCREMENT</td>
<td>CDF_INFO</td>
<td>CDF_INT1</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>CDF_INT4</td>
<td>CDF_INT8</td>
</tr>
<tr>
<td>CDF_INTERNAL_ERROR</td>
<td>CDF_MAJORITY</td>
<td>CDF_MAX_DIMS</td>
</tr>
<tr>
<td>CDF_MAX_PARMS</td>
<td>CDF_MIN_DIMS</td>
<td>CDF_NAME</td>
</tr>
<tr>
<td>CDF_NAME_TRUNC</td>
<td>CDF_NEGtoPOSfp0_MODE</td>
<td>CDF_NUMATTRS</td>
</tr>
<tr>
<td>CDF_NUMgATTRS</td>
<td>CDF_NUMrVARS</td>
<td>CDF_NUMvATTRS</td>
</tr>
<tr>
<td>CDF_OK</td>
<td>CDF_PATHNAME_LEN</td>
<td>CDF_READ_ERROR</td>
</tr>
<tr>
<td>CDF_READONLY_MODE</td>
<td>CDF_REAL4</td>
<td>CDF_REAL8</td>
</tr>
<tr>
<td>CDF_RELEASE</td>
<td>CDF_SAVE_ERROR</td>
<td>CDF_SCRATCHDIR</td>
</tr>
<tr>
<td>CDF_STATUS</td>
<td>CDF_STATUSTEXT_LEN</td>
<td>CDF_TIME_TT2000</td>
</tr>
<tr>
<td>CDF_UCHAR</td>
<td>CDF_UINT1</td>
<td>CDF_UINT2</td>
</tr>
<tr>
<td>CDF_UINT4</td>
<td>CDF_VAR_NAME_LEN</td>
<td>CDF_VAR_NAME_LEN256</td>
</tr>
<tr>
<td>CDF_VERSION</td>
<td>CDF_WARN</td>
<td>CDF_WRITE_ERROR</td>
</tr>
<tr>
<td>CDF_zMODE</td>
<td>CDFwithSTATS</td>
<td>CHECKSUM</td>
</tr>
<tr>
<td>CHECKSUM_ERROR</td>
<td>CHECKSUM_NOT_ALLOWED</td>
<td>CLOSE</td>
</tr>
<tr>
<td>COLUMN_MAJOR</td>
<td>COMPRESS_CACHESIZE</td>
<td>COMPRESSION_ERROR</td>
</tr>
<tr>
<td>COMPRESS_ERROR</td>
<td>CONFIRM</td>
<td>CORRUPTED_V2_CDF</td>
</tr>
<tr>
<td>CORRUPTED_V3_CDF</td>
<td>CREATE</td>
<td>CURgENTRY_EXISTENCE</td>
</tr>
<tr>
<td>CURrENTRY_EXISTENCE</td>
<td>CURzENTRY_EXISTENCE</td>
<td>DATATYPE_MISMATCH</td>
</tr>
<tr>
<td>DATATYPE_SIZE</td>
<td>DECOMPRESS_ERROR</td>
<td>DECIMATION_DECODING</td>
</tr>
<tr>
<td>DECOMPRESSION_ERROR</td>
<td>DECSTATION_DECODING</td>
<td>DEFAULT_BYTE_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_CHAR_PADVALUE</td>
<td>DEFAULT_DOUBLE_PADVALUE</td>
<td>DEFAULT_EPOCH_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_FLOAT_PADVALUE</td>
<td>DEFAULT_INT1_PADVALUE</td>
<td>DEFAULT_INT2_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_INT4_PADVALUE</td>
<td>DEFAULT_INT8_PADVALUE</td>
<td>DEFAULT_REAL4_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_REAL8_PADVALUE</td>
<td>DEFAULT_TT2000_PADVALUE</td>
<td>DEFAULT_UDCHAR_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_UINT1_PADVALUE</td>
<td>DEFAULT_UINT2_PADVALUE</td>
<td>DEFAULT_UINT4_PADVALUE</td>
</tr>
<tr>
<td>DEFAULT_TT2000_PADVALUE</td>
<td>DELETE</td>
<td>DID_NOT_COMPRESS</td>
</tr>
<tr>
<td>DUMMY_TT2000_VALUE</td>
<td>EMPTY_COMPRESSED_CDF</td>
<td>END_OF_VAR</td>
</tr>
<tr>
<td>EPOCH_STRING_LEN</td>
<td><a href="http://localhost:8080/cdfdocs/gsfc/nssdc/cdf/Entry.html">http://localhost:8080/cdfdocs/gsfc/nssdc/cdf/Entry.html</a> (2 of 9) [7/12/2011 7:01:16 PM]</td>
<td></td>
</tr>
</tbody>
</table>
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Entry <code>create</code> (Attribute myAttribute, long id, long dataType, java.lang.Object data)</td>
<td>Creates a new global or variable attribute entry.</td>
</tr>
<tr>
<td>void <code>delete</code> ()</td>
<td>Deletes this entry.</td>
</tr>
<tr>
<td>java.lang.Object <code>getData</code> ()</td>
<td>Gets the data for this entry.</td>
</tr>
<tr>
<td>long <code>getDataType</code> ()</td>
<td>Gets the CDF data type of this entry.</td>
</tr>
<tr>
<td>long <code>getID</code> ()</td>
<td>Gets the ID of this entry.</td>
</tr>
<tr>
<td>java.lang.String <code>getName</code> ()</td>
<td>Gets the name of this entry.</td>
</tr>
<tr>
<td>long <code>getNumElements</code> ()</td>
<td>Gets the number of elements in this entry.</td>
</tr>
<tr>
<td>void <code>putData</code> (long dataType, java.lang.Object data)</td>
<td>Put the entry data into the CDF.</td>
</tr>
</tbody>
</table>
### Method Detail

#### create

```java
public static Entry create(Attribute myAttribute,
    long id,
    long dataType,
    java.lang.Object data)
    throws CDFException
```

Creates a new global or variable attribute entry. One can create as many global and variable entries as needed. The following example creates four entries for the global attribute "Project":

```java
Attribute project  = Attribute.create(cdf, "Project", GLOBAL_SCOPE);
Entry.create(project, 0, CDF_CHAR, "Project name: IMAGE");
Entry.create(project, 1, CDF_CHAR, "Description 1");
Entry.create(project, 2, CDF_CHAR, "Description 2");
```

The following example creates a variable attribute entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```java
Variable longitude = cdf.getVariable("Longitude");
Attribute validMin = Attribute.create(cdf, "VALIDMIN",
    VARIABLE_SCOPE);
Entry.create(validMin, longitude.getID(), CDF_INT2,
    new Short((short)10));
```

OR

```java
longitude.putEntry(validMin, CDF_INT2, new Short((short)180));
```
Parameters:

myAttribute - the attribute to which this entry belongs

id - the entry id

dataType - the CDF data type for this entry that should be one of the following:

- CDF_BYTE - 1-byte, signed integer
- CDF_CHAR - 1-byte, signed character
- CDF_INT1 - 1-byte, signed integer
- CDF_UCHAR - 1-byte, unsigned character
- CDF_UINT1 - 1-byte, unsigned integer
- CDF_INT2 - 2-byte, signed integer
- CDF_UNIT2 - 2-byte, unsigned integer
- CDF_INT4 - 4-byte, signed integer
- CDF(UINT4) - 4-byte, unsigned integer
- CDF_INT8 - 8-byte, signed integer
- CDF_REAL4 - 4-byte, floating point
- CDF_FLOAT - 4-byte, floating point
- CDF_REAL8 - 8-byte, floating point
- CDF_DOUBLE - 8-byte, floating point
- CDF_EPOCH - 8-byte, floating point
- CDF_EPOCH16 - 2*8-byte, floating point
- CDF_TIME_TT2000 - 8-byte, signed integer

data - the entry data to be added

Returns:

newly created attribute entry

Throws:

CDFException - if there is a problem creating an entry

---

delete

define public void delete() throws CDFException

Deletes this entry.

Specified by:

delete in interface CDFObject

Throws:

CDFException - if there is a problem deleting this entry
**getDataType**

```java
class Entry
{
    public long getDataType()
    {
        Gets the CDF data type of this entry. See the description of the create method for the CDF data types supported by the CDF library.

        Returns:
        the CDF data type of this entry
    }
}
```

**getNumElements**

```java
class Entry
{
    public long getNumElements()
    {
        Gets the number of elements in this entry. For CDF_CHAR, it returns the number of characters stored.

        Entry data        Number of elements
        ----------        ------------------
        10                1
        20.8              1
        10 20 30          3
        20.8 20.9         2
        "Upper Limits"    12

        Returns:
        the number of elements stored in this entry
    }
}
```

**getData**

```java
class Entry
{
    public java.lang.Object getData()
    {
        Gets the data for this entry.

        Returns:
        the data for this entry
    }
}
```
**getID**

```java
public long getID()
```

Gets the ID of this entry.

**Returns:**
the ID/number of this entry

---

**getName**

```java
public java.lang.String getName()
```

Gets the name of this entry. Since an entry doesn't have its own name, the string representation of this entry ID is returned.

This method overrides the getName() method defined in the Java Object class. If this method is called explicitly or implicitly (i.e. just the entry name by itself), it returns the string representation of the entry ID.

**Specified by:**
getName in interface CDFObject

**Returns:**
string representation of this attribute entry ID

---

**rename**

```java
public void rename(java.lang.String name)
    throws CDFException
```

This method is here as a placeholder since the Entry class implements the CDFObject interface that includes "rename".

**Specified by:**
rename in interface CDFObject

**Parameters:**
name -- not applicable

**Throws:**
CDFException -- not applicable
**updateDataSpec**

```java
public void updateDataSpec(long dataType,
                            long numElements)
    throws CDFException
```

Update the data specification (data type and number of elements) of the entry.

**Throws:**

CDFException

---

**putData**

```java
public void putData(long dataType,
                    java.lang.Object data)
    throws CDFException
```

Put the entry data into the CDF.

**Throws:**

CDFException
public class Epoch
extends java.lang.Object
implements CDFConstants

Example:

// Get the milliseconds to Aug 5, 1990 at 5:00
double ep = Epoch.compute(1990, 8, 5, 5, 0, 0, 0);
//Get the year, month, day, hour, minutes, seconds, milliseconds for ep
long times[] = Epoch.breakdown(ep);
for (int i=0;i<times.length;i++)
    System.out.print(times[i]+" ");
System.out.println();
// Printout the epoch in various formats
System.out.println(Epoch.encode(ep));
System.out.println(Epoch.encode1(ep));
System.out.println(Epoch.encode2(ep));
System.out.println(Epoch.encode3(ep));
System.out.println(Epoch.encode4(ep));
// Print out the date using format
String format = " , at ";
System.out.println(Epoch.encoder(ep,format));
## Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

| AHUFF_COMPRESSION | ALPHAOSF1_DECODING | ALPHAOSF1_ENCODING | ALPHAVMSd_DECODING | ALPHAVMSd_ENCODING | ALPHAVMSq_DECODING | ALPHAOSF1_DECODING | ALPHAVMSd_ENCODING | ALPHAVMSq_ENCODING | Attributes | ATTR_EXISTENCE | ATTR_EXISTS | ATTR_MAXgENTRY | ATTR_MAXrENTRY | ATTR_MAXzENTRY | ATTR_NAME | ATTR_NAME_TRUNC | ATTR_NUMBER | ATTR_NUMgENTRIES | ATTR_NUMrENTRIES | ATTR_NUMzENTRIES | ATTR_SCOPE | BACKWARD | BACKWARDFILEoff | BACKWARDFILEon | BAD_ALLOCATE_RECS | BAD_ARGUMENT | BAD_ATTR_NAME | BAD_ATTRNUM | BAD_BLOCKING_FACTOR | BAD_CACHE_SIZE | BAD_CDF_EXTENSION | BAD_CDF_ID | BAD_CDF_NAME | BAD_CDFSTATUS | BAD_CHECKSUM | BAD_COMPRESSION_PARM | BAD_DATA_TYPE | BAD_DECODING | BAD_DIM_COUNT | BAD_DIM_INDEX | BAD_DIM_INTERVAL | BAD_DIM_SIZE | BAD_ENCODING | BAD_ENTRY_NUM | BAD_FNC_OR_ITEM | BAD_FORMAT | BAD_INITIAL_RECS | BAD_MAJOR | BAD_MALLOC | BAD_NEGtoPOSfp0_MODE | BAD_NUM_DIMS | BAD_NUM_ELEMS | BAD_NUM_VARS | BAD_SCOPE | BAD_SCRATCH_DIR | BAD_SPARSEARRAYS_PARM | BAD_VAR_NAME | BAD_VAR_NUM | BAD_zMODE | CANNOT_ALLOCATE_RECORDS | CANNOT_CHANGE | CANNOT_COMPRESS | CANNOT_COPY | CANNOT_SPARSEARRAYS | CANNOT_SPARSERECORDS | CDF | CDF_ACCESS | CDF_ATTR_NAME_LEN | CDF_ATTR_NAME_LEN256 | CDF_BYTE | CDF_CACHE_SIZE | CDF_CHAR | CDF_CHECKSUM | CDF_CLOSE_ERROR | CDF_COMPRESSION | CDF_COPYRIGHT | CDF_COPYRIGHT_LEN | CDF_CREATE_ERROR | CDF_DECODING | CDF_DELETE_ERROR | CDF_DOUBLE | CDF_ENCODING | CDF_EPOCH | CDF_EPOCH16 | CDF_EXISTS | CDF_FLOAT | CDF_FORMAT | CDF_INCREMENT | CDF_INFO | CDF_INT1 | CDF_INT2 | CDF_INT4 | CDF_INT8 | CDF_INTERNAL_ERROR | CDF_MAJOR | CDF_MAX_DIMS | CDF_MAX_PARMS | CDF_MIN_DIMS | CDF_NAME | CDF_NAME_TRUNC | CDF_NEGtoPOSfp0_MODE | CDF_NUMATTRS | CDF_NUMqATTRS | CDF_NUMrVARS | CDF_NUMvATTRS | CDF_NUMzVARS | CDF_OK | CDF_OPEN_ERROR | CDF_PATHNAME_LEN | CDF_READ_ERROR | CDF_READONLY_MODE | CDF_REAL4 | CDF_REAL8 | CDF_RELEASE | CDF_SAVE_ERROR | CDF_SCRATCHDIR | CDF_STATUS | CDF_STATUSTEXT_LEN | CDF_TIME_TT2000 | CDF_UCHAR | CDF_UINT1 | CDF_UINT2 | CDF_UINT4 | CDF_VAR_NAME_LEN | CDF_VAR_NAME_LEN256 | CDF_VERSION |
CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_,
CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR,
COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF,
CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_,
CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_,
DECOMPRESSION_ERROR, DECOMPRESSION_DECODING, DECOMPRESSION_ENCODING,
DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE,
DEFAULT_EPOCH_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,
DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE,
DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE,
DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE,
DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE,
DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, DUMMY_TT2000_VALUE,
EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN,
EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND,
EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN,
EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND,
EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FORCED_PARAMETER, gENTRY_,
gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_,
gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_,
GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE,
GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING,
HP_ENCODING, HUFF_COMPRESSION, IBM_PC_OVERFLOW, IBMPC_DECODING,
IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD,
ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE,
ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_,
LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_,
MAC_DECODING, MAC.Encoding, MD5_CHECKSUM, MULTI_FILE,
MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off,
NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING,
NeXT.Encoding, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM,
NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS,
NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARESERECORDS,
NO_STATUS_Selected, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY,
NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED,
NOVARY, NULL_, OPEN_, OPTIMAL.Encoding_TREES, OTHER_CHECKSUM,
PAD_SPARESERECORDS, PPC_DECODING, PPC_ENCODING,
PRECEEDING_RECORDS_ALLOCATED, PREV_SPARESERECORDS, PUT_.

Epoch

READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, RLE_COMPRESSION, RLE_OF_ZEROS, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECOVERY_, rVAR_reservePERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERCORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCHCREATE_ERROR, SCRATCHDELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGi_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_TIME_ERROR, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VARDELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_MAXALLOCRECS_, zVAR_NINDEXRECORDS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNumer_, zVAR_RECORDS_, zVAR_RECOVERY_, zVAR_reservePERCENT_, zVAR_SEQDATA_, zVARs_CNTS_,
Constructor Summary

`Epoch()`

Method Summary

<table>
<thead>
<tr>
<th>static long[]</th>
<th><code>breakdown(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Breaks an EPOCH value down into its component parts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><code>compute(long year, long month, long day, long hour, long minute, long second, long msec)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computes an EPOCH value based on its component parts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encode(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encode1(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encode2(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encode3(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encode4(double epoch)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time, ISO8601 string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><code>encodex(double epoch, java.lang.String formatString)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string using the specified format.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><code>parse(java.lang.String inString)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><code>parse1(java.lang.String inString)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
<tr>
<td>static double parse2(java.lang.String inString)</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
This function parses an input date/time string and returns an EPOCH value.

<table>
<thead>
<tr>
<th>static double parse3(java.lang.String inString)</th>
</tr>
</thead>
</table>
This function parses an input date/time string and returns an EPOCH value.

<table>
<thead>
<tr>
<th>static double parse4(java.lang.String inString)</th>
</tr>
</thead>
</table>
This function parses an input date/time string and returns an EPOCH value.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait

**Constructor Detail**

**Epoch**

public Epoch()

**Method Detail**

**parse**

public static double parse(java.lang.String inString)

throws CDFException

This function parses an input date/time string and returns an EPOCH value. The format must be exactly as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

**Format:** dd-mmm-yyy hh:mm:ss.mmm

**Examples:** 1-Apr-1990 03:05:02.000
10-Oct-1993 23:45:49.999
The expected format is the same as that produced by encodeEPOCH.

**Parameters:**
- inString - the epoch in string representation

**Returns:**
- the value of the epoch represented by inString

**Throws:**
- CDFException - if a bad epoch value is passed in inString

---

**parse1**

```java
public static double parse1(java.lang.String inString)
throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be exactly as shown below. Note that if there are less than 7 digits after the decimal point, zeros (0's) are assumed for the missing digits.

**Format:**    yyyymmdd.ttttttt

**Examples:**  19950508.0000000
19671231.58      (== 19671213.5800000)

The expected format is the same as that produced by encodeEPOCH1.

**Parameters:**
- inString - the epoch in string representation

**Returns:**
- the value of the epoch represented by inString

**Throws:**
- CDFException - if a bad epoch value is passed in inString

---

**parse2**

```java
public static double parse2(java.lang.String inString)
throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be
Epoch

exactly as shown below.

Format:     yyyyymmdhhmmss
Examples:   19950508000000
            196712311235959

The expected format is the same as that produced by encodeEPOCH2.

Parameters:
inString - the epoch in string representation

Returns:
the value of the epoch represented by inString

Throws:
  CDFException - if a bad epoch value is passed in inString

---

class parse3

public static double parse3(java.lang.String inString)
  throws CDFException

This function parses an input date/time string and returns an EPOCH value. The format must be
exactly as shown below.

Format:     yyyy-mm-ddThh:mm:ss.cccZ
Examples:   1990-04-01T03:05:02.000Z
            1993-10-10T23:45:49.999Z

The expected format is the same as that produced by encodeEPOCH3.

Parameters:
inString - the epoch in string representation

Returns:
the value of the epoch represented by inString

Throws:
  CDFException - if a bad epoch value is passed in inString
parse4

public static double parse4(java.lang.String inString)
    throws CDFException

This function parses an input date/time string and returns an EPOCH value. The format must be an ISO8601 and is exactly as shown below.

Format:  yyyy-mm-ddThh:mm:ss.ccc
Examples:  1990-04-01T03:05:02.000
             1993-10-10T23:45:49.999

The expected format is the same as that produced by encodeEPOCH3.

Parameters:
    inString - the epoch in string representation

Returns:
    the value of the epoch represented by inString

Throws:
    CDFException - if a bad epoch value is passed in inString

encode

public static java.lang.String encode(double epoch)

Converts an EPOCH value into a readable date/time string.

Format:  dd-mmm-yyyy hh:mm:ss.ccc
Examples:  01-Apr-1990 03:05:02.000
             10-Oct-1993 23:45:49.999

This format is the same as that expected by parse.

Parameters:
    epoch - the epoch value

Returns:
    A string representation of the epoch
**encode1**

```java
public static java.lang.String encode1(double epoch)
```

Converts an EPOCH value into a readable date/time string.

**Format:** yyyyymmd.dttttttt

**Examples:** 19900401.3658893
19611231.0000000

This format is the same as that expected by parse1.

**Parameters:**
- **epoch** - the epoch value

**Returns:**
- A string representation of the epoch

---

**encode2**

```java
public static java.lang.String encode2(double epoch)
```

Converts an EPOCH value into a readable date/time string.

**Format:** yyyyymmdhhmmss

**Examples:** 19900401235959
19611231000000

This format is the same as that expected by parse2.

**Parameters:**
- **epoch** - the epoch value

**Returns:**
- A string representation of the epoch
**encode3**

public static java.lang.String **encode3**(double epoch)

Converts an EPOCH value into a readable date/time string.

Format:    yyyy-mm-ddThh:mm:ss.cccZ
Examples:  1990-04-01T03:05:02.000Z
           1993-10-10T23:45:49.999Z

This format is the same as that expected by parse3.

**Parameters:**

   epoch - the epoch value

**Returns:**

A string representation of the epoch

---

**encode4**

public static java.lang.String **encode4**(double epoch)

Converts an EPOCH value into a readable date/time, ISO8601 string.

Format:    yyyy-mm-ddThh:mm:ss.ccc
Examples:  1990-04-01T03:05:02.000
           1993-10-10T23:45:49.999

This format is the same as that expected by parse3.

**Parameters:**

   epoch - the epoch value

**Returns:**

A string representation of the epoch

---

**encodex**
public static java.lang.String encode(double epoch, java.lang.String formatString)

Converts an EPOCH value into a readable date/time string using the specified format. See the C Reference Manual section 8.7 for details

Parameters:
  epoch - the epoch value
  formatString - a string representing the desired format of the epoch

Returns:
  A string representation of the epoch according to formatString

compute

public static double compute(long year, long month, long day, long hour, long minute, long second, long msec)
  throws CDFException

Computes an EPOCH value based on its component parts.

Parameters:
  year - the year
  month - the month
  day - the day
  hour - the hour
  minute - the minute
  second - the second
  msec - the millisecond

Returns:
  the epoch value

Throws:
  CDFException - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.
public static long[] breakdown(double epoch)

Breaks an EPOCH value down into its component parts.

Parameters:
epoch - the epoch value to break down

Returns:
an array containing the epoch parts:

<table>
<thead>
<tr>
<th>Index</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>year</td>
</tr>
<tr>
<td>1</td>
<td>month</td>
</tr>
<tr>
<td>2</td>
<td>day</td>
</tr>
<tr>
<td>3</td>
<td>hour</td>
</tr>
<tr>
<td>4</td>
<td>minute</td>
</tr>
<tr>
<td>5</td>
<td>second</td>
</tr>
<tr>
<td>6</td>
<td>msec</td>
</tr>
</tbody>
</table>

Overview  Package  Class  Tree  Deprecated  Index  Help
PREV CLASS  NEXT CLASS  FRAMES  NO FRAMES  All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
gsfc.nssdc.cdf.util

Class Epoch16

double[] epoch16 = new double[2];
double ep = Epoch16.compute(1990, 8, 5, 5, 0, 0, 0, 0, 0, 0, epoch16);
// Get the year, month, day, hour, minutes, seconds, milliseconds,
// microseconds, nanoseconds and picoseconds for epoch16
long times[] = Epoch16.breakdown(epoch16);
for (int i=0;i<times.length;i++)
    System.out.print(times[i]+" ");
System.out.println();
// Printout the epoch in various formats
System.out.println(Epoch16.encode(epoch16));
System.out.println(Epoch16.encode1(epoch16));
System.out.println(Epoch16.encode2(epoch16));
System.out.println(Epoch16.encode3(epoch16));
System.out.println(Epoch16.encode4(epoch16));
// Print out the date using format
String format = " , at ";
System.out.println(Epoch16.encodex(epoch16,format));

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd.Encoding, ALPHAVMSg_DECODING, ALPHAVMSg.Encoding, ALPHAVMSi_DECODING, ALPHAVMSi.Encoding, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTRIB_NAME, BAD_ATTRIB_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJOR, BAD_MAJOR_, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS, CANNOT_SPARSE RECORDS, CDF_, CDF_ACCESS_, CDF_ATTRIB_NAME_LEN, CDF_ATTRIB_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_MAJOR, CDF_MAJOR_, CDF_MAX_DIMS, CDF_MAX_PARMs, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRs, CDF_NUMgATTRs, CDF_NUMrVARS_, CDF_NUMvATTRs, CDF_NUMzVARS, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4_
Epoch16

NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY,
NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF,
NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_,
OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS,
PPC_DECODING, PPC_ENCODING, PRECEENDING_RECORDS_ALLOCATED,
PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE,
READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_,
rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_,
RLE_COMPRESSION, RLE_OF_ZEROS, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_,
rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_,
rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_,
rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_,
rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_,
rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_,
rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_,
rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_,
rVAR_RECVARY_, rVAR_reservePERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_,
rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_,
rVARs_DIMCOUNTS_, rVARsDIMINDICES_, rVARs_DIMINTERVALS_,
rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_,
rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNNUMBER_, SAVE_,
SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR,
SCRATCH_WRITE_ERROR, SELECT_, SGi_DECODING, SGi_ENCODING,
SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED,
STAGE_CACHESIZE_, STATUS_TEXT_, SUN_DECODING, SUN_ENCODING,
TOO_MANY_PARMS, TOO_MANY_VARS, TT2000_0_STRING_LEN,
TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN,
TT2000_TIME_ERROR, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS,
UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon,
VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR,
VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR,
VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE,
VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA_, zENTRY_,
zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_,
zENTRY_NAME_, zENTRY_NUMELEMS_, zMODEoff, zMODEon1, zMODEon2, zVAR_,
zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_,
zVAR_ALLOCATERECS_, zVAR_COMPRESSION_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_,
zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_,
### Constructor Summary

**Epoch16**

### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long[]</td>
<td><code>breakdown(java.lang.Object epoch)</code></td>
<td>Breaks an EPOCH16 value down into its component parts.</td>
</tr>
<tr>
<td>static double</td>
<td><code>compute(long year, long month, long day, long hour, long minute, long second, long msec, long usec, long nsec, long psec, java.lang.Object epoch)</code></td>
<td>Computes an EPOCH16 value based on its component parts.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><code>encode(java.lang.Object epoch)</code></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><code>encode1(java.lang.Object epoch)</code></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><code>encode2(java.lang.Object epoch)</code></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><code>encode3(java.lang.Object epoch)</code></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><code>encode4(java.lang.Object epoch)</code></td>
<td>Converts an EPOCH16 value into a readable date/time, ISO8601 string.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>static java.lang.String <strong>encodex</strong>(java.lang.Object epoch, java.lang.String formatString)</td>
<td>Converts an EPOCH16 value into a readable date/time string using the specified format.</td>
<td></td>
</tr>
<tr>
<td>static java.lang.Object <strong>parse</strong>(java.lang.String inString)</td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
<td></td>
</tr>
<tr>
<td>static java.lang.Object <strong>parse1</strong>(java.lang.String inString)</td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
<td></td>
</tr>
<tr>
<td>static java.lang.Object <strong>parse2</strong>(java.lang.String inString)</td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
<td></td>
</tr>
<tr>
<td>static java.lang.Object <strong>parse3</strong>(java.lang.String inString)</td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
<td></td>
</tr>
<tr>
<td>static java.lang.Object <strong>parse4</strong>(java.lang.String inString)</td>
<td>This function parses an input date/time, ISO8601 string and returns an EPOCH16 value.</td>
<td></td>
</tr>
</tbody>
</table>

**Methods inherited from class java.lang.Object**

- equals, getClass, hashCode, notify, notifyAll, toString, wait, wait

**Constructor Detail**

**Epoch16**

public **Epoch16**()
public static java.lang.Object parse(java.lang.String inString) throws CDFException

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

Format: dd-mmm-yyyy hh:mm:ss.ccc.mmm.nnn.ppp
Examples: 1-Apr-1990 03:05:02.000.000.000.000
          10-Oct-1993 23:45:49.999.999.999.999

The expected format is the same as that produced by encode.

Parameters:
  inString - the epoch in string representation

Returns: the value of the epoch represented by inString

Throws: CDFException - if a bad epoch value is passed in inString

parse1

public static java.lang.Object parse1(java.lang.String inString) throws CDFException

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below. Note that if there are less than 15 digits after the decimal point, zeros (0's) are assumed for the missing digits.

Format: yyyymmd.ttttttttttttttt
Examples: 19950508.000000000000000
          19671231.58 (== 19671213.580000000000000)

The expected format is the same as that produced by encode1.

Parameters:
  inString - the epoch in string representation

Returns:
the value of the epoch represented by inString

Throws:
CDFException - if a bad epoch value is passed in inString

parse2

public static java.lang.Object parse2(java.lang.String inString)
throws CDFException

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below.

Format:         yyyyymmddhhmmss
Examples:       19950508000000
                19671231235959

The expected format is the same as that produced by encode2.

Parameters:
inString - the epoch in string representation

Returns:
the value of the epoch represented by inString

Throws:
CDFException - if a bad epoch value is passed in inString

parse3

public static java.lang.Object parse3(java.lang.String inString)
throws CDFException

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below.

Format:         yyyy-mm-ddThh:mm:ss.ccc.mmm.nnn.pppZ
Examples:       1990-04-01T03:05:02.000.000.000.000Z
                1993-10-10T23:45:49.999.999.999.999Z
The expected format is the same as that produced by encode3.

**Parameters:**
inString - the epoch in string representation

**Returns:**
the value of the epoch represented by inString

**Throws:**
[CDFException](#) - if a bad epoch value is passed in inString

---

**parse4**

```java
throws CDFException
```

This function parses an input date/time, ISO8601 string and returns an EPOCH16 value. The format must be exactly as shown below.

**Format:**
```
yyyy-mm-ddThh:mm:ss.cccmmmnnnppp
```

**Examples:**
```
1990-04-01T03:05:02.000000000000
1993-10-10T23:45:49.999999999999
```

The expected format is the same as that produced by encode3.

**Parameters:**
inString - the epoch in string representation

**Returns:**
the value of the epoch represented by inString

**Throws:**
[CDFException](#) - if a bad epoch value is passed in inString

---

**encode**

```java
public static java.lang.String encode(java.lang.Object epoch)
```
Converts an EPOCH16 value into a readable date/time string.

**Format:**
dd-mm-yyyy hh:mm:ss.ccc.mmm.nnn.ppp

**Examples:**
01-Apr-1990 03:05:02.000.000.000.000
10-Oct-1993 23:45:49.999.999.999.999

This format is the same as that expected by parse.

**Parameters:**
- epoch - the epoch value

**Returns:**
A string representation of the epoch

---

**encode1**

```java
public static java.lang.String encode1(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

**Format:**
yyyyymmdd.ttttttttttttttt

**Examples:**
19900401.365889312341234
19611231.000000000000000

This format is the same as that expected by parse1.

**Parameters:**
- epoch - the epoch value

**Returns:**
A string representation of the epoch

---

**encode2**

```java
public static java.lang.String encode2(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.
Epoch16

Format:         yyyymmddhhmss
Examples:       19900401235959
                19611231000000

This format is the same as that expected by parse2.

Parameters:
    epoch - the epoch value

Returns:
    A string representation of the epoch

---

encode3

public static java.lang.String encode3(java.lang.Object epoch)

Converts an EPOCH16 value into a readable date/time string.

Format:         yyyy-mm-ddThh:mm:ss.ccc.mmm.nnn.pppZ
Examples:       1990-04-01T03:05:02.000.000.000.000Z
                1993-10-10T23:45:49.999.999.999.999Z

This format is the same as that expected by parse3.

Parameters:
    epoch - the epoch value

Returns:
    A string representation of the epoch

---

encode4

public static java.lang.String encode4(java.lang.Object epoch)

Converts an EPOCH16 value into a readable date/time, ISO8601 string.

Format:         yyyy-mm-ddThh:mm:ss.cccmmmnnnppp
Examples: 1990-04-01T03:05:02.000000000
1993-10-10T23:45:49.999999999

This format is the same as that expected by parse4.

Parameters:
    epoch - the epoch value

Returns:
    A string representation of the epoch

encodex

public static java.lang.String encodex(java.lang.Object epoch,
                                       java.lang.String formatString)

Converts an EPOCH16 value into a readable date/time string using the specified format. See the C Reference Manual section 8.7 for details

Parameters:
    epoch - the epoch value
    formatString - a string representing the desired format of the epoch

Returns:
    A string representation of the epoch according to formatString

compute

public static double compute(long year,
                              long month,
                              long day,
                              long hour,
                              long minute,
                              long second,
                              long msec,
                              long usec,
                              long nsec,
                              long psec,
                              java.lang.Object epoch)
Epoch16

Throws: `CDFException`

Computes an EPOCH16 value based on its component parts.

**Parameters:**
- `year` - the year
- `month` - the month
- `day` - the day
- `hour` - the hour
- `minute` - the minute
- `second` - the second
- `msec` - the milliseconds
- `usec` - the microseconds
- `nsec` - the nanoseconds
- `psec` - the picoseconds

**Returns:**
- the epoch value

**Throws:**
- `CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

---

breakdown

public static long[] `breakdown`(java.lang.Object epoch)

Breaks an EPOCH16 value down into its component parts.

**Parameters:**
- `epoch` - the epoch value to break down

**Returns:**
- an array containing the epoch parts:

<table>
<thead>
<tr>
<th>Index</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>year</td>
</tr>
<tr>
<td>1</td>
<td>month</td>
</tr>
<tr>
<td>2</td>
<td>day</td>
</tr>
<tr>
<td>3</td>
<td>hour</td>
</tr>
<tr>
<td>4</td>
<td>minute</td>
</tr>
<tr>
<td>5</td>
<td>second</td>
</tr>
</tbody>
</table>
6 msec
7 usec
8 nsec
9 psec
The Epoch class is a Java wrapper to the CDF epoch handling routines. See Chapter 8 of the CDF C Reference Manual Version 2.6 for details

**Example:**

```java
// Get the milliseconds to Aug 5, 1990 at 5:00
double ep = Epoch.compute(1990, 8, 5, 0, 0, 0);
//Get the year, month, day, hour, minutes, seconds, milliseconds for ep
long times[] = Epoch.breakdown(ep);
for (int i=0; i < times.length; i++)
```

```
## Constructor Summary

**EpochNative()**

## Method Summary

<table>
<thead>
<tr>
<th>static long[]</th>
<th><strong>breakdown</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors EPOCHbreakdown from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><strong>compute</strong>(long year, long month, long day, long hour, long minute, long second, long msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors computeEPOCH from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCH from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode1</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCH1 from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode2</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCH2 from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode3</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCH3 from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode4</strong>(double epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCH4 from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encodex</strong>(double epoch, java.lang.String format)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors encodeEPOCHx from the CDF library.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><strong>parse</strong>(java.lang.String sEpoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mirrors parseEPOCH from CDF library.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>static double parse1</td>
<td>Mirrors parseEPOCH from CDF library.</td>
</tr>
<tr>
<td>static double parse2</td>
<td>Mirrors parseEPOCH from CDF library.</td>
</tr>
<tr>
<td>static double parse3</td>
<td>Mirrors parseEPOCH from CDF library.</td>
</tr>
<tr>
<td>static double parse4</td>
<td>Mirrors parseEPOCH from CDF library.</td>
</tr>
</tbody>
</table>

**Methods inherited from class java.lang.Object**

- equals
- getClass
- hashCode
- notify
- notifyAll
- toString
- wait
- wait

### Constructor Detail

```
public EpochNative()
```
compute

public static double compute(long year,
                   long month,
                   long day,
                   long hour,
                   long minute,
                   long second,
                   long msec)

MIRRORS computeEPOCH from the CDF library. See Section 8.1 of the

breakdown

public static long[] breakdown(double epoch)

MIRRORS EPOCHbreakdown from the CDF library. See Section 8.2 of the
**encode**

```java
public static java.lang.String encode(double epoch)
```

MIRRORS `encodeEPOCH` FROM THE CDF LIBRARY. SEE SECTION 8.3 OF THE CDF C REFERENCE MANUAL VERSION 2.6 FOR DETAILS

---

**encode1**

```java
public static java.lang.String encode1(double epoch)
```

MIRRORS `encodeEPOCH1` FROM THE CDF LIBRARY. SEE SECTION 8.4 OF THE CDF C REFERENCE MANUAL VERSION 2.6 FOR DETAILS

---
**encode2**

```java
public static java.lang.String encode2(double epoch)
```

Mirrors `encodeEPOCH2` from the CDF library. See Section 8.5 of the CDF C Reference Manual Version 2.6 for details.

---

**encode3**

```java
public static java.lang.String encode3(double epoch)
```

Mirrors `encodeEPOCH3` from the CDF library. See Section 8.6 of the CDF C Reference Manual Version 2.6 for details.
EpochNative

**encode4**

```java
public static java.lang.String encode4(double epoch)
```

Mirrors encodeEPOCH4 from the CDF library. See Section 8.6 of the CDF C Reference Manual Version 2.6 for details.

**encodex**

```java
public static java.lang.String encodex(double epoch, java.lang.String format)
```

Mirrors encodeEPOCHx from the CDF library. See Section 8.7 of the CDF C Reference Manual Version 2.6 for details.
public static double parse(java.lang.String sEpoch)

Mirrors parseEPOCH from CDF library. See Section 8.8 of the CDF C Reference Manual Version 2.6 for details

parse1

public static double parse1(java.lang.String sEpoch)

Mirrors parseEPOCH from CDF library. See Section 8.9 of the CDF C Reference Manual Version 2.6 for details

parse2

public static double parse2(java.lang.String sEpoch)
Mirrors parseEPOCH from CDF library. See Section 8.10 of the CDF C Reference Manual Version 2.6 for details

parse3

public static double parse3(java.lang.String sEpoch)

Mirrors parseEPOCH from CDF library. See Section 8.11 of the CDF C Reference Manual Version 2.6 for details

parse4

public static double parse4(java.lang.String sEpoch)

Mirrors parseEPOCH from CDF library. See Section 8.11 of the CDF C Reference Manual Version 2.6 for details
gsfc.nssdc.cdf

Class CDFException

```
java.lang.Object
  └── java.lang.Throwable
      └── java.lang.Exception
          └── gsfc.nssdc.cdf.CDFException
```

All Implemented Interfaces:

CDFConstants, java.io.Serializable

---

public class CDFException

extends java.lang.Exception
implements CDFConstants

This class defines the informational, warning, and error messages that can arise from CDF operations.

See Also:

Serialized Form

---

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants
CDFException

AHUFF_COMPRESSION, ALPHASF1_DECODING, ALPHASF1_ENCODING,
ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING,
ALPHAVMSg_ENCODING, ALPHAVSI_DECODING, ALPHAVSI_ENCODING, ATTR_,
ATTR_EXISTENCE, ATTR_EXISTS, ATTR_MAXgENTRY, ATTR_MAXrENTRY,
ATTR_MAXzENTRY, ATTR_NAME, ATTR_NAME_TRUNC, ATTR_NUMBER,
ATTR_NUMgENTRIES, ATTR_NUMrENTRIES, ATTR_NUMzENTRIES, ATTR_SCOPE_,
BACKWARD, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS,
BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR,
BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME,
BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE,
BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL,
BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT,
BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE,
BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_VARS, BAD_READONLY_MODE,
BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR,
BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE,
CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_COPY,
CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_,
CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_,
CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_,
CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_,
CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16,
CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO, CDF_INT1,
CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_MAJORITY_,
CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC,
CDF_NEGtoPOSfp0_MODE, CDF_NUMATTRS, CDF_NUMgATTRS, CDF_NUMrVARS,
CDF_NUMzATTRS, CDF_NUMzVARS, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN,
CDF_READ_ERROR, CDF_READONLY_MODE, CDF_REAL4, CDF_REAL8, CDF_RELEASE_,
CDF_SAVE_ERROR, CDF_SCRATCHDIR, CDF_STATUS, CDF_STATUSTEXT_LEN,
CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4,
CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN,
CDF_WRITE_ERROR, CDF_zMODE, CDFwithSTATS, CHECKSUM, CHECKSUM_ERROR,
CHECKSUM_NOT_ALLOWED, CLOSE, COLUMN_MAJOR, COMPRESS_CACHESIZE_,
COMPRESS_ERROR, CONFIRM, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_,
CURgENTRY_EXISTENCE, CURrENTRY_EXISTENCE, CURzENTRY_EXISTENCE_,
DATATYPE_MISMATCH, DATATYPE_SIZE, DECOMPRESSION_ERROR,
DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE,
DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE,
DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE,
DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE,
DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE,
DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, DUMMY TT2000_VALUE, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FORCED_PARAMETER, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPEXCONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_INMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECs_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYs_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECs_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELs_, rVAR_nINDEXRECORDs_, rVAR_NUMallocRECs_, rVAR_NUMBER_, rVAR_NUMELEMs_, rVAR_NUMRECs_, rVAR_PADVALUE_, rVAR_RECORDs_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEODATA_, rVAR_SEOPos_, rVAR_SPARSEARRAYs_, rVAR_SPARSERECORDs_, rVARs_CACHESIZE_, rVARs_DIMCOUNTs_, rVARs_DIMINDICES_, rVARs_DIMINTERVALs_, rVARs_DIMSIZEs_, rVARs_MAXREC_, rVARs_NUMDIMs_, rVARs_RECeOUNT_, rVARs_RECDATA_
**Constructor Summary**

`CDFException` (long statusCode)

Takes a status code and throws a `CDFException` with the message that corresponds to the status code that is passed in.

`CDFException` (long statusCode, java.lang.String where)

Takes a status code and throws a `CDFException` with the message that corresponds to the status code that is passed in.

`CDFException` (java.lang.String message)

Takes a text message from the calling program and throws a `CDFException`.

**Method Summary**
### Constructor Detail

#### CDFException

**public CDFException(java.lang.String message)**

Takes a text message from the calling program and throws a CDFException.

**Parameters:**
- `message` - the message to be thrown with CDFException

#### CDFException

**public CDFException(long statusCode)**

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.

**Parameters:**
- `statusCode` - the CDF statusCode to be thrown
public CDFException(long statusCode, java.lang.String where)

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in. It also specifies where (which routine) the problem was.

Parameters:
- statusCode - the CDF statusCode to be thrown
- where - the place (routine/method) where the problem occurred

Method Detail

getCurrentStatus

public long getCurrentStatus()

Gets the status code that caused CDFException. This method comes in handy when there are times one may want to examine the cause of the CDFException and determine whether to continue or not.

try {
    ...

    } catch (CDFException e) {
        if (e.getCurrentStatus() == NO_SUCH_VAR) {
            Variable latitude = Variable.create(cdf, "Latitude", CDF_INT1, numElements, numDims, dimSizes, recVary, dimVary);

            ...
        }
        else {
            System.out.println ("StatusCode = "+e.getCurrentStatus());
            e.printStackTrace();
        }
    }
}
**Returns:**
the status code that caused CDFException

---

**get_status_msg**

public static java.lang.String `get_status_msg`(long statusCode)

Get the status text message for the given status code.

**Parameters:**

statusCode - the status code from which the status text is retrieved

**Returns:**

the status text message for the given status code
The `Variable` class defines a CDF variable.

Notes: Since the CDF Java API always uses `zMODE = 2`, all variables are by default, `zVariables`.

Version:

1.0, 2.0 03/18/05 Selection of current CDF and variable are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called.

See Also:

`Attribute`, `Entry`
Variable

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING,
ALPHAVMsd_DECODING, ALPHAVMsd_ENCODING, ALPHAVMsq_DECODING,
ALPHAVMsq_ENCODING, ALPHAVMsI_DECODING, ALPHAVMsI_ENCODING, ATTR_
ATTR_EXISTENCE, ATTR_EXISTS, ATTR_MAXgENTRY, ATTR_MAXrENTRY,
ATTR_MAXzENTRY, ATTR_NAME, ATTR_NAME_TRUNC, ATTR_NUMBER,
ATTR_NUMgENTRIES, ATTR_NUMrENTRIES, ATTR_NUMzENTRIES, ATTR_SCOPE,
BACKWARD, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS,
BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR,
BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS,
BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING,
BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORIY,
BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS,
BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL,
BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM,
BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, CANNOT_ALLOCATE_RECORDS,
CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,
CANNOT_SPARSERECORDS, CDF__, CDF_ACCESS__, CDF_ATTR_NAME_LEN,
CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE__, CDF_CHAR, CDF_CHECKSUM__,
CDF_CLOSE_ERROR, CDF_COMPRESSION__, CDF_COPYRIGHT__, CDF_COPYRIGHT_LEN,
CDF_CREATE_ERROR, CDF_DECODING__, CDF_DELETE_ERROR, CDF_DOUBLE,
CDF_ENCODING__, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT__,
CDF_INCREMENT__, CDF_INFO__, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8,
CDF_INTERNAL_ERROR, CDF_MAJORIY__, CDF_MAX_DIMS, CDF_MAX_PARMS,
CDF_MIN_DIMS, CDF_NAME__, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE__,
CDF_NUMATTRS__, CDF_NUMgATTRS__, CDF_NUMrVARS__, CDF_NUMvATTRS__,
CDF_NUMzVARS__, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR,
CDF_READONLY_MODE__, CDF_REAL4, CDF_REAL8, CDF_RELEASE__, CDF_SAVE_ERROR,
CDF_SCRATCHDIR__, CDF_STATUS__, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000,
CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN,
CDF_VAR_NAME_LEN256, CDF_VERSION__, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE__,
CDFwithSTATS__, CHECKSUM__, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE__,
COLUMN_MAJOR__, COMPRESS_CACHESIZE__, COMPRESSION_ERROR, CONFIRM__,
CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE__, CURgENTRY_EXISTENCE__,
CURrENTRY_EXISTENCE__, CURzENTRY_EXISTENCE__, DATATYPE_MISMATCH, 
DATATYPE_SIZE__, DECOMPRESSION_ERROR, DECASTATION_DECODING, 
DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, 
DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_FLOAT_PADVALUE, 
DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, 
DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, 
DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, 
DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE__, DID_NOT_COMPRESS,
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>allocateBlock</strong>(long firstRec, long lastRec)</td>
<td>Allocates a range of records for this variable.</td>
</tr>
<tr>
<td><strong>allocateRecords</strong>(long num0toRecords)</td>
<td>Allocates a number of records, starting from record number 0.</td>
</tr>
<tr>
<td><strong>checkPadValueExistence</strong>()</td>
<td>Checks if the pad value has been defined for this variable.</td>
</tr>
<tr>
<td><strong>concatenateDataRecords</strong>(Variable destVar)</td>
<td>Concatenates this variable's data records to the destination variable.</td>
</tr>
<tr>
<td><strong>confirmCacheSize</strong>()</td>
<td>Gets the number of 512-byte cache buffers defined for this variable.</td>
</tr>
<tr>
<td><strong>confirmPadValue</strong>()</td>
<td>Checks the existence of an explicitly specified pad value for the current z variable.</td>
</tr>
<tr>
<td><strong>confirmReservePercent</strong>()</td>
<td>Gets the reserve percentage set for this variable.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **copy** | (CDF destCDF, java.lang.String varName)  
Copies this variable into a new variable and puts it into the designated CDF file. |
| **copy** | (java.lang.String varName)  
Copies this variable to a new variable. |
| **copyDataRecords** | (Variable destVar)  
Copies this variable's data to the destination variable. |
| **create** | (CDF myCDF, java.lang.String varName, long dataType,  
long numElements, long numDims, long[] dimSizes,  
long recVary, long[] dimVarys)  
Creates a variable. |
| **delete** | ()  
Deletes this variable. |
| **deleteRecords** | (long firstRec, long lastRec)  
Deletes a range of records from this variable. |
| **duplicate** | (CDF destCDF, java.lang.String varName)  
Duplicates this variable and put it into the designated CDF file. |
| **duplicate** | (java.lang.String varName)  
Duplicates this variable to a new variable. |
| **getAllocatedFrom** | (long recNum)  
Inquires the next allocated record at or after a given record for this variable. |
| **getAllocatedTo** | (long firstRec)  
Inquires the last allocated record (before the next unallocated record) at or after a given record for this variable. |
| **getAttributes** | ()  
Returns the variable attributes that are associated with this variable. |
| **getBlockingFactor** | ()  
Gets the blocking factor for this variable. |
| **getCompression** | ()  
Gets the string representation of the compression type and parameters set for this variable. |
| **getCompressionParms** | ()  
Sets the compression parameters of this variable. |
| **getCompressionPct** | ()  
Gets the compression percentage rate of this variable. |
| **getCompressionType** | ()  
Gets the compression type of this variable. |
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long getDataType()</code></td>
<td>Gets the CDF data type of this variable.</td>
</tr>
<tr>
<td><code>long[] getDimSizes()</code></td>
<td>Gets the dimensions size of this variable.</td>
</tr>
<tr>
<td><code>long[] getDimVariances()</code></td>
<td>Gets the dimension variances for this variable.</td>
</tr>
<tr>
<td><code>java.lang.Object getEntryData(java.lang.String attrName)</code></td>
<td>Gets the attribute entry data for this variable.</td>
</tr>
<tr>
<td><code>java.lang.Object getHyperData(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals)</code></td>
<td>Reads one or more values from the current z variable.</td>
</tr>
<tr>
<td><code>CDFData getHyperDataObject(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals)</code></td>
<td>Reads one or more values from the current z variable.</td>
</tr>
<tr>
<td><code>longgetID()</code></td>
<td>Gets the ID of this variable.</td>
</tr>
<tr>
<td><code>long getMaxAllocatedRecord()</code></td>
<td>Gets the maximum allocated record number for this variable.</td>
</tr>
<tr>
<td><code>long getMaxWrittenRecord()</code></td>
<td>Gets the last written record number, beginning with 0.</td>
</tr>
<tr>
<td><code>CDF getMyCDF()</code></td>
<td>Gets the CDF object to which this variable belongs.</td>
</tr>
<tr>
<td><code>java.lang.String getName()</code></td>
<td>Gets the name of this variable.</td>
</tr>
<tr>
<td><code>long getNumAllocatedRecords()</code></td>
<td>Gets the number of records allocated for this variable.</td>
</tr>
<tr>
<td><code>long getNumDims()</code></td>
<td>Gets the number of dimensions for this variable.</td>
</tr>
<tr>
<td><code>long getNumElements()</code></td>
<td>Gets the number of elements for this variable.</td>
</tr>
<tr>
<td><code>long getNumWrittenRecords()</code></td>
<td>Gets the number of records physically written (not allocated) for this variable.</td>
</tr>
<tr>
<td><code>java.lang.Object getPadValue()</code></td>
<td>Gets the pad value set for this variable.</td>
</tr>
<tr>
<td><code>java.lang.Object getRecord(long recNum)</code></td>
<td>Gets a single record from this variable.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getRecordObject(long recNum)</code></td>
<td>Get a single record of data from this variable.</td>
</tr>
<tr>
<td><code>getRecordsObject(long recNum, long numRecs)</code></td>
<td>Get a number of records of data from this variable.</td>
</tr>
<tr>
<td><code>getRecVariance()</code></td>
<td>Gets the value of record variance.</td>
</tr>
<tr>
<td><code>getScalarData()</code></td>
<td>Gets the scalar data from a non-record varying 0-dimensional variable.</td>
</tr>
<tr>
<td><code>getScalarData(long recNum)</code></td>
<td>Get the scalar data from a record varying 0-dimensional variable.</td>
</tr>
<tr>
<td><code>getScalarDataObject()</code></td>
<td>Get the scalar data from a non-record varying 0-dimensional variable.</td>
</tr>
<tr>
<td><code>getScalarDataObject(long recNum)</code></td>
<td>Get the scalar data from this record varying 0-dimensional variable.</td>
</tr>
<tr>
<td><code>getSingleData(long recNum, long[] indices)</code></td>
<td>Gets a single data value.</td>
</tr>
<tr>
<td><code>getSingleDataObject(long recNum, long[] indices)</code></td>
<td>Gets a single data object from this variable.</td>
</tr>
<tr>
<td><code>getSparseRecords()</code></td>
<td>Gets the sparse record type for this variable.</td>
</tr>
<tr>
<td><code>putEntry(Attribute attr, long dataType, java.lang.Object data)</code></td>
<td>Creates an attribute entry for this variable.</td>
</tr>
<tr>
<td><code>putEntry(java.lang.String attrName, long dataType, java.lang.Object data)</code></td>
<td>Creates an attribute entry for this variable.</td>
</tr>
<tr>
<td><code>putHyperData(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals, java.lang.Object data)</code></td>
<td>Writes one or more values from the current z variable.</td>
</tr>
<tr>
<td><code>putRecord(long recNum, java.lang.Object data)</code></td>
<td>Adds a single record to a record-varying variable.</td>
</tr>
<tr>
<td><code>putRecord(java.lang.Object data)</code></td>
<td>Adds a single record to a non-record-varying variable.</td>
</tr>
<tr>
<td><code>putScalarData(long recNum, java.lang.Object data)</code></td>
<td>Adds a scalar data to this variable (of 0 dimensional).</td>
</tr>
</tbody>
</table>
### Methods

- **putScalarData** *(java.lang.Object data)*
  Adds a scalar data to this variable (of 0 dimensional).

- **putSingleData** *(long recNum, long[] indices, java.lang.Object data)*
  Adds a single data value to this variable.

- **rename** *(java.lang.String newName)*
  Renames the current variable.

- **selectCacheSize** *(long cacheSize)*
  Sets the number of 512-byte cache buffers to be used.

- **selectReservePercent** *(long reservePercent)*
  Sets the reserve percentage to be used for this variable.

- **setBlockingFactor** *(long blockingFactor)*
  Sets the blocking factor for this variable.

- **setCompression** *(long cType, long[] cParms)*
  Sets the compression type and parameters for this variable.

- **setDimVariances** *(long[] dimVariances)*
  Sets the dimension variances for this variable.

- **setInitialRecords** *(long nRecords)*
  Sets the number of records to be written initially for this variable.

- **setPadValue** *(java.lang.Object padValue)*
  Sets the pad value for this variable.

- **setRecVariance** *(long recVariance)*
  Sets the record variance for this variable.

- **setSparseRecords** *(long sparseRecords)*
  Sets the sparse record type for this variable.

- **toString** *
  Gets the name of this variable.

- **updateDataSpec** *(long dataType, long numElements)*
  Update the data specification (data type and number of elements) of the variable.

---

### Methods inherited from class java.lang.Object

- equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
public static Variable create(CDF myCDF,
        java.lang.String varName,
        long dataType,
        long numElements,
        long numDims,
        long[] dimSizes,
        long recVary,
        long[] dimVarys)
    throws CDFException

Creates a variable.

The following example creates a variable called "Longitude" that is scalar (non-array) and record-varying:

    longitude = Variable.create(cdf, "Longitude", CDF_INT2,
        1L, 0L, new long [] {1},
        VARY,
        new long [] {NOVARY});

The following example creates a variable called "TestData" whose data is 2-dimensional (3 x 2), record variance is TURE, and dimension variances are TRUE.

    data = Variable.create(cdf, "TestData", CDF_INT2,
        1L, 2L, new long [] {3,2},
        VARY,
        new long [] {VARY, VARY});

Parameters:

    myCDF - the CDF to which this variable belongs

    varName - the name of the variable to create

    dataType - the CDF data type for this variable that should be one of the following:
        ■ CDF_BYTE - 1-byte, signed integer
        ■ CDF_CHAR - 1-byte, signed character
        ■ CDF_INT1 - 1-byte, signed integer
        ■ CDF_UCHAR - 1-byte, unsigned character
        ■ CDF_UINT1 - 1-byte, unsigned integer
        ■ CDF_INT2 - 2-byte, signed integer
        ■ CDF_UNIT2 - 2-byte, unsigned integer
        ■ CDF_INT4 - 4-byte, signed integer
        ■ CDF_UINT4 - 4-byte, unsigned integer
Variable

- CDF_INT8 - 8-byte, signed integer
- CDF_REAL4 - 4-byte, floating point
- CDF_FLOAT - 4-byte, floating point
- CDF_REAL8 - 8-byte, floating point
- CDF_DOUBLE - 8-byte, floating point
- CDF_EPOCH - 8-byte, floating point
- CDF_EPOCH16 - 2*8-byte, floating point
- CDF_TIME_TT2000 - 8-byte, signed integer

numElements - for CDF_CHAR and CDF_UCHAR this is the string length, 1 otherwise

numDims - the dimensionality

dimSizes - The dimension sizes. An array of length numDims indicating the size of each dimension

recVary - the record variance that should be either VARY or NOVARY

dimVarys - The dimension variance(s). Each dimension variance should be either VARY or NOVARY.

Returns:
newly created Variable object

Throws:
CDFException - if there is a problem creating a variable

```java
public void delete()
throws CDFException
```

Deletes this variable.

Specified by:
delete in interface CDFObject

Throws:
CDFException - if there was an error deleting this variable

rename
public void rename(java.lang.String newName) throws CDFException

Renames the current variable.

Specified by:
   rename in interface CDFObject

Parameters:
   newName - the new variable name

Throws:
   CDFException - if there was a problem renaming this variable

---

copy

copy

public Variable copy(java.lang.String varName) throws CDFException

Copies this variable to a new variable. This method only copies the metadata associated with this variable. The duplicate method in this class should be used if the user wants to copy a variable with data and metadata.

Parameters:
   varName - the name of the variable to copy this variable into

Returns:
   newly copied variable

Throws:
   CDFException - if there was a problem copying a variable

---

copy

copy

public Variable copy(CDF destCDF, java.lang.String varName) throws CDFException

Copies this variable into a new variable and puts it into the designated CDF file. This method only copies the metadata associated with this variable. The duplicate method in this class should be used if the user wants to copy a variable with data and metadata.
### Parameters:
- destCDF - the destination CDF into which copy this variable
- varName - the new variable name

### Returns:
newly copied variable

### Throws:
- CDFException - if there was a problem copying a variable

```java
public Variable duplicate(java.lang.String varName)
throws CDFException
```

Duplicates this variable to a new variable.

**Note:** This copies everything from the existing variable to a new variable. It includes the metadata associated with this variable, all data records as well as other information such as blocking factor/compression/sparseness/pad value.

### Parameters:
- varName - the name of the variable to duplicate this variable into

### Returns:
newly duplicated variable

### Throws:
- CDFException - if there was a problem duplicating a variable

```java
public Variable duplicate(CDF destCDF, java.lang.String varName)
throws CDFException
```

Duplicates this variable and put it into the designated CDF file.

**Note:** This copies everything from the current variable to a new variable. It includes the metadata associated with this variable, all data records as well as other information such as blocking factor/compression/sparseness/pad value.
Variable

Parameters:
- destCDF - the destination CDF to duplicate this variable into
- varName - the name of the variable to duplicate this variable into

Returns:
- newly duplicated variable

Throws:
- CDFException - if there was a problem duplicating a variable

---

**copyDataRecords**

```java
public void copyDataRecords(Variable destVar)
    throws CDFException
```

Copies this variable's data to the destination variable.

**Note:** This copies data records from the current variable to the destination variable. The metadata associated with the destination variable will be not changed.

The current CDF file MUST be saved first (by calling the save() method) before 'copying/duplicating data records' operation is performed. Otherwise the program will either fail or produce undesired results.

**Parameters:**
- destVar - the destination variable to copy data into

**Throws:**
- CDFException - if there was a problem copying data records

---

**concatenateDataRecords**

```java
public void concatenateDataRecords(Variable destVar)
    throws CDFException
```

Concatenates this variable's data records to the destination variable.

**Note:** This copies only the data records from the current variable to the destination variable. The metadata associated with the destination variable will be not changed.

**Parameters:**
**destVar** - the destination variable to copy data records into

**Throws:**
- [CDFException](#) - if there was a problem copying data records

---

### getEntryData

```java
public java.lang.Object getEntryData(java.lang.String attrName) throws CDFException
```

Gets the attribute entry data for this variable.

The following examples retrieves the 'Longitude' variable entry for the attribute VALIDMIN:

```java
Variable var = cdf.getVariable("Longitude");
float longitude = (float) var.getEntryData("VALIDMIN");
```

**Parameters:**
- `attrName` - the name of the attribute to get entry data from

**Returns:**
- the attribute entry data for this variable

**Throws:**
- [CDFException](#) - if there was a problem getting entry data

---

### getSingleData

```java
public java.lang.Object getSingleData(long recNum, long[] indices) throws CDFException
```

Gets a single data value. This method is useful for extracting a specific item among many items.

Let's assume that variable TestData is defined to be 1-dimensional array that has 3 elements in it. The following example extracts the last element from the second record:

```java
Variable var = cdf.getVariable("TestData");
int data = (int) var.getSingleData(1L, new long [] {2});
```
Let's assume that variableTestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns) array. The following example extracts the first element of the second row from the first record:

```java
Variable var = cdf.getVariable("TestData");
int data = (int) var.getSingleData(0L, new long [] {1,0});
```

**Parameters:**
- recNum - the record number to retrieve data from
- indices - the index, within a record, to extract data from

**Returns:**
- extracted single data value

**Throws:**
- `CDFException` - if there was a problem extracting data

---

**getSingleDataObject**

```java
public CDFData getSingleDataObject(long recNum,
        long[] indices)
        throws CDFException
```

Gets a single data object from this variable. The value read is put into an CDFData object. This method is identical to the getSingleData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

**Parameters:**
- recNum - the record number to retrieve data from
- indices - the index, within a record, to extract data from

**Returns:**
- CDFData object containing the requested data

**Throws:**
- `CDFException` - if there was a problem extracting data
getRecord

public java.lang.Object getRecord(long recNum)
        throws CDFException

    Gets a single record from this variable.

    Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The
    following example extracts an entire record (containing 6 elements) from the first record from a variable of
data type CDF_INT4:

    Variable var = cdf.getVariable("TestData");
    int[][] data = (int [][]) var.getRecord(0L);

    However, if a dimensional variable with all indices being invariant, e.g., 2-dimensional (1x1), the retrieved
    object will be different. (Since the variable has only one data value per record, it is preferred to be defined
    as an 0-dim, rather.). The object is not an array, instead, a single Java class item, e.g., Integer, Double,
    Short, etc. The following example extracts an record from the first record of a variable, 2-dim (1x1), with
data type CDF_INT2:

    Variable var = cdf.getVariable("TestVar");
    short data = ((Short) var.getRecord(0L)).shortValue();

    Parameters:
        recNum - the record number to retrieve data from

    Returns:
        the requested data record

    Throws:
        CDFException - if there was a problem getting a record

getRecordObject

public CDFData getRecordObject(long recNum)
        throws CDFException

    Get a single record of data from this variable. The values read are put into an CDFData object. This method
    is identical to the getRecord method except that the extracted data is encapsulated inside the CDFData
    object along with other information such as record number, record count, record interval, dimension
    indices, dimension counts, and dimension intervals.
Parameters:
   recNum - the record number to retrieve data from

Returns:
   CDOObject containing the requested data record

Throws:
   CDFException - if there was a problem getting a record

getRecordsObject

public CDFData getRecordsObject(long recNum,  
    long numRecs)  
    throws CDFException

Get a number of records of data from this variable. The values read are put into an CDFData object.

Parameters:
   recNum - the record number to start to retrieve data from

   numRecs - the number of records to retrieve

Returns:
   CDOObject containing the requested data record(s)

Throws:
   CDFException - if there was a problem getting the record(s)

getScalarData

public java.lang.Object getScalarData()  
    throws CDFException

Gets the scalar data from a non-record varying 0-dimensional variable.

Returns:
   the variable data from this variable

Throws:
getScalarData

```java
public java.lang.Object getScalarData(long recNum)
        throws CDFException
```

Get the scalar data from a record varying 0-dimensional variable.

**Parameters:**
- `recNum` - The record number to retrieve data from

**Returns:**
- the variable data from this variable

**Throws:**
- `CDFException` - if there was a problem getting data

getScalarDataObject

```java
public CDFData getScalarDataObject()
        throws CDFException
```

Get the scalar data from a non-record varying 0-dimensional variable. This method is identical to the getScalarData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

**Returns:**
- the variable data from this variable

**Throws:**
- `CDFException` - if there was a problem getting data

getScalarDataObject

```java
public CDFData getScalarDataObject(long recNum)
        throws CDFException
```


Get the scalar data from this record varying 0-dimensional variable. This method is identical to the getScalarData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

**Parameters:**
- `recNum` - the record number to retrieve data from

**Returns:**
- the variable data from this variable

**Throws:**
- `CDFException` - if there was a problem getting data

---

### getHyperData

```java
public java.lang.Object getHyperData(long recNum,
    long recCount,  
    long recInterval,  
    long[] dimIndices,
    long[] dimCounts,  
    long[] dimIntervals)
    throws CDFException
```

Reads one or more values from the current z variable. The values are based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals.

Let's assume that variable `TestData` is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example extracts the entire record (containing 6 elements) from the first, second, and third records:

```java
Variable var = cdf.getVariable("TestData");
int[][][] data = (int [][][]) var.getHyperData (0L, 3L, 1L,
    new long[] {0, 0},   
    new long[] {3, 2},   
    new long[] {1, 1});
```

The following example will extract the entire record from the first record:

```java
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 1L, 1L,
    new long[] {0, 0},   
    new long[] {0, 0},
```
Note: it returns a 2-dimensional object as only one record is involved. The following example will extract the second row from the first, and third records:

```java
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 3L, 2L,
        new long[] {1, 0},
        new long[] {1, 2},
        new long[] {1, 1});
```

The following example will extract the first column from the first and second records:

```java
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 2L, 1L,
        new long[] {0, 0},
        new long[] {3, 1},
        new long[] {1, 1});
```

**Parameters:**
- `recNum` - the record number at which data search begins
- `recCount` - the number of records to read
- `recInterval` - the number of records to skip between reads
- `dimIndices` - the dimension index within a record at which data search begins
- `dimCounts` - the number of elements to read from `dimIndices`
- `dimIntervals` - the number of elements to skip between reads

**Returns:**
the variable data specified by `recNum`, `recCount`, `recInterval`, `dimIndices`, `dimCounts`, and `dimIntervals`

**Throws:**
- `CDFException` - if there was a problem getting data
getHyperDataObject

```java
public CDFData getHyperDataObject(long recNum,
   long recCount,
   long recInterval,
   long[] dimIndices,
   long[] dimCounts,
   long[] dimIntervals)
   throws CDFException
```

Reads one or more values from the current z variable. The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals. The values read are put into an CDFData object.

**Parameters:**
- `recNum` - the record number at which data search begins
- `recCount` - the number of records to read
- `recInterval` - the number of records to skip between reads
- `dimIndices` - the dimension index within a record at which data search begins
- `dimCounts` - the number of elements to read from `dimIndices`
- `dimIntervals` - the number of elements to skip between reads

**Returns:**
CDFData object that contains the variable data specified by `recNum`, `recCount`, `recInterval`, `dimIndices`, `dimCounts`, and `dimIntervals` as well as the information passed to this method plus the number of dimensions and the number of elements for this variable.

**Throws:**
- `CDFException` - if there was a problem getting data

---

putEntry

```java
public void putEntry(java.lang.String attrName,
   long dataType,
   java.lang.Object data)
   throws CDFException
```

Creates an attribute entry for this variable.

The following example creates a variable entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```
    Variable longitude = cdf.getVariable("Longitude");
```
longitude.putEntry("VALIDMIN", CDF_INT2, new Short((short)180));

**Parameters:**
- attrName - the attribute to which this attribute entry is attached
- dataType - the CDF data type of the entry data - see the description of the create method in this class for a list of the CDF data types supported
- data - the attribute entry data to be added

**Throws:**
- CDFException - if a problem occurs putting an entry

**See Also:**
- Attribute, Entry

---

**putEntry**

```java
public void putEntry(Attribute attr, long dataType, java.lang.Object data)
throws CDFException
```

Creates an attribute entry for this variable. The following example creates a variable entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```java
Variable longitude = cdf.getVariable("Longitude");
Attribute validMin = Attribute.create(cdf, "VALIDMIN", VARIABLE_SCOPE);
Entry.create(validMin, longitude.getID(), CDF_INT2, new Short((short)10));
```

OR

```java
longitude.putEntry(validMin, CDF_INT2, new Short((short)180));
```

**Parameters:**
- attr - the attribute to which this attribute entry is attached
- dataType - the CDF data type of the entry data - see the description of the create method in this class for a list of the CDF data types supported
- data - the attribute entry data to be added
Throws:

- **CDFException** - if a problem occurs putting an entry

See Also:

- **Attribute**, **Entry**

---

### putSingleData

```java
public CDFData putSingleData(long recNum,
       long[] indices,
       java.lang.Object data)
throws CDFException
```

Adds a single data value to this variable. This method is used to specify a particular element in a record (if a record is comprised of multiple elements). If a record contains 3 elements, the following example will write the second element to record number 0, leaving the first and third elements unwritten.

```java
longitude = cdf.getVariable("Longitude");
longitude.putSingleData(0L, new long[] {1}, new Short((short)200));
or
longitude.putSingleData(0L, new long[] {1}, longitudeData[1]);
```

**Parameters:**

- `recNum` - the record number to which this data belongs
- `indices` - the index (location) in the specified record
- `data` - the data to be added

**Returns:**

- CDFData object containing the user specified data

**Throws:**

- **CDFException** - if there was an error writing data

---

### putScalarData

```java
public CDFData putScalarData(long recNum,
                             java.lang.Object data)
throws CDFException
```

---

http://localhost:8080/cdfdocs/gsfc/nssdc/cdf/Variable.html (23 of 41) [7/12/2011 7:01:24 PM]
Variable

Adds a scalar data to this variable (of 0 dimensional). This method should be used if a variable is defined as record-varying and non-array. The following example will write data to record number 0.

```java
longitude = cdf.getVariable("Longitude");
longitude.putScalarData(0L, new Short((short)200));
or
longitude.putScalarData(0L, longitudeData[0]);
```

Parameters:
- recNum - the record number to which this data belongs
- data - the data to be added

Returns:
- CDFData object containing the user specified data

Throws:
- CDFException - if there was an error writing data

putScalarData

public CDFData putScalarData(java.lang.Object data) throws CDFException

Adds a scalar data to this variable (of 0 dimensional). This method should be used if a variable is defined as non-record-varying and non-array. Note that there'll be only one record exist if a variable is defined as non-record-varying. The following example will write data to record number 0.

```java
longitude = cdf.getVariable("Longitude");
longitude.putScalarData(new Short((short)200));
or
longitude.putScalarData(longitudeData[0]);
```

Parameters:
- data - the data to be added

Returns:
- CDFData object containing the user specified data

Throws:
**putRecord**

```java
public CDFData putRecord(long recNum, java.lang.Object data) throws CDFException
```

Adds a single record to a record-varying variable. This method should be used if a record contains one or more elements.

The following example adds a scalar data to record number 0:

```java
longitude = cdf.getVariable("Longitude");
longitude.putRecord(0L, new Short((short)200));
```

The following example adds multiple elements (array) to record number 0:

```java
short [] longitudeData = {10, 20, 30};
longitude = cdf.getVariable("Longitude");
longitude.putRecord(0L, longitudeData);
```

**Parameters:**
- `recNum` - the record number to which this data belongs
- `data` - the data to be added

**Returns:**
CDFData object containing the user specified data

**Throws:**
- `CDFException` - if there was a problem writing data
**putRecord**

`public CDFData putRecord(java.lang.Object data) throws CDFException`  

Adds a single record to a non-record-varying variable. This method should be used if a record contains one element or multiple elements.

The following example adds a scalar data to record number 0:

```java
longitude = cdf.getVariable("Longitude");
longitude.putRecord(new Short((short)200));
```

The following example adds multiple elements (array) to record number 0:

```java
short [] longitudeData = {10, 20, 30};
longitude = cdf.getVariable("Longitude");
longitude.putRecord(longitudeData);
```

**Parameters:**
- `data` - the data to be added

**Returns:**
- CDFData object containing the user specified data

**Throws:**
- `CDFException` - if there was a problem writing data

**putHyperData**

`public CDFData putHyperData(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts,`
long[] dimIntervals,
java.lang.Object data)
throws CDFException

Writes one or more values from the current z variable. The values are written based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals. The values read are put into an CDFData object. Although this method returns a CDFData object, it is not necessary to capture the return value to a CDFData variable.

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example writes the entire record (containing 6 elements) to the first, second, and third records:

```java
long [][] testData = {{{10,20},{30,40},{50, 60}},
                    {{15,25},{45,55},{75, 85}}
                    {{90,95},{96,97},{2147483648L,4294967295L}}};
testData.putHyperData (0L, 3L, 1L,
                        new long[] {0, 0},
                        new long[] {3, 2},
                        new long[] {1, 1});
```

The following example will write the first two rows of testData to the first, third, and fifth records:

```java
testData.putHyperData (0L, 3L, 2L,
                        new long[] {0, 0},
                        new long[] {2, 2},
                        new long[] {1, 1});
```

**Parameters:**
- `recNum`: the record number at which data write begins
- `recCount`: the number of records to write
- `recInterval`: the number of records to skip between writes
- `dimIndices`: the dimension index within a record at which data write begins
- `dimCounts`: the number of elements to write from `dimIndices`
- `dimIntervals`: the number of elements to skip between writes
- `data`: the data to be written
**Returns:**
CDFData object that contains the variable data specified by recNum, recCount, recInterval, dimIndices, dimCounts, and dimIntervals as well as the information passed to this method plus the number of dimensions and the number of elements for this variable.

**Throws:**
[CDFException](#) - if there was a problem writing data

---

**getMyCDF**

```java
public CDF getMyCDF()
```

Gets the CDF object to which this variable belongs.

**Returns:**
the CDF object to which this variable belongs

---

**getCompressionType**

```java
public long getCompressionType()
```

Gets the compression type of this variable.

**Returns:**
the compression type of this variable

---

**getCompressionPct**

```java
public long getCompressionPct()
```

Gets the compression percentage rate of this variable.

**Returns:**
the compression percentage rate of this variable

---

**getCompressionParms**
public long[] `getCompressionParms()`

Sets the compression parameters of this variable. This is only applicable for the GZIP compression method.

**Returns:**
the compression parameters of this variable

---

**setCompression**

public void `setCompression` (long cType, long[] cParms) throws `CDFException`

Sets the compression type and parameters for this variable.

**Parameters:**
cType - the compression type
cParms - the compression parameters that go with cType

**Throws:**
`CDFException` - if a problem occurs setting compression type and parameters

---

**getCompression**

public java.lang.String `getCompression`() throws `CDFException`

Gets the string representation of the compression type and parameters set for this variable.

**Returns:**
the string representation of the compression type and parameters for this variable

**Throws:**
`CDFException` - if a problem occurs getting the compression type and parameters

---

**getNumDims**
public long getNumDims()

    Gets the number of dimensions for this variable.

    **Returns:**
        the number of dimensions for this variable

---

**getDimSizes**

public long[] getDimSizes()

    Gets the dimensions size of this variable.

    **Returns:**
        the dimension size of this variable

---

**getNumElements**

public long getNumElements()

    Gets the number of elements for this variable. For CDF_CHAR and CDF_UCHAR this is the number of characters in the string. For all other types this defaults to 1.

    **Returns:**
        the number of elements for this variable

---

**getName**

public java.lang.String getName()

    Gets the name of this variable.

    **Specified by:**
        getName in interface CDFObject

    **Returns:**
        the name of this variable
**getID**

public long **getID**()

    Gets the ID of this variable.

    **Returns:**
    the ID of this variable

---

**toString**

public java.lang.String **toString**()

    Gets the name of this variable.

    **Overrides:**
    toString in class java.lang.Object

    **Returns:**
    the name of this variable

---

**setRecVariance**

public void **setRecVariance**(long recVariance)
    throws CDFException

    Sets the record variance for this variable.

    **Parameters:**
    recVariance - the record variance that should be either VARY or NOVARY.

    **Throws:**
    CDFException - if a problem occurs setting the record variance

---

**getRecVariance**

public boolean **getRecVariance**()
Variable

Gets the value of record variance.

**Returns:**
True if this variable is record varying, False otherwise

---

**setDimVariances**

```java
public void setDimVariances(long[] dimVariances)
    throws CDFException
```

Sets the dimension variances for this variable.

**Parameters:**
dimVariances - the dimension variances for this variable

**Throws:**
CDFException - if a problem occurs setting the dimension variances

---

**getDimVariances**

```java
public long[] getDimVariances()
```

Gets the dimension variances for this variable.

**Returns:**
the dimension variances for this variable

---

**getDataType**

```java
public long getDataType()
```

Gets the CDF data type of this variable.

**Returns:**
the CDF data type of this variable
deleteRecords

public void deleteRecords(long firstRec,
    long lastRec)
    throws CDFException

Deletes a range of records from this variable.

Parameters:
    firstRec - the first record to be deleted
    lastRec - the last record to be deleted

Throws:
    CDFException - if a problem occurs deleting records

allocateBlock

public void allocateBlock(long firstRec,
    long lastRec)
    throws CDFException

Allocates a range of records for this variable.

Parameters:
    firstRec - the first record to be allocated
    lastRec - the last record to be allocated

Throws:
    CDFException - if a problem occurs allocating records

allocateRecords

public void allocateRecords(long num0toRecords)
    throws CDFException

Allocates a number of records, starting from record number 0.

Parameters:
    num0toRecords - the number of records to be allocated
Throws:
   [CDFException] - if a problem occurs allocating records

---

getNumWrittenRecords

public long getNumWrittenRecords()
   throws [CDFException]

Gets the number of records physically written (not allocated) for this variable.

Returns:
   the number of records written physically

Throws:
   [CDFException] - if a problem occurs getting the number of records written physically

---

getMaxWrittenRecord

public long getMaxWrittenRecord()
   throws [CDFException]

Gets the last written record number, beginning with 0.

Returns:
   the last written record number

Throws:
   [CDFException] - if a problem occurs getting the last written record number

---

getNumAllocatedRecords

public long getNumAllocatedRecords()
   throws [CDFException]

Gets the number of records allocated for this variable.

Returns:
the number of records allocated

Throws:
   CDFException - if a problem occurs getting the number of records allocated

---

### getMaxAllocatedRecord

```java
public long getMaxAllocatedRecord() throws CDFException
```

Gets the maximum allocated record number for this variable.

**Returns:**
   the maximum allocated record number

**Throws:**
   CDFException - if a problem occurs getting the maximum allocated record number

---

### setPadValue

```java
public void setPadValue(java.lang.Object padValue) throws CDFException
```

Sets the pad value for this variable. This pad value is used, when storing data, for undefined values.

**Parameters:**
   padValue - the pad value to be used for undefined values

**Throws:**
   CDFException - if a problem occurs setting the pad value

---

### checkPadValueExistence

```java
public boolean checkPadValueExistence() throws CDFException
```

Checks if the pad value has been defined for this variable. While the getPadValue() method always returns
a pad value, it may simply be the default pad value (albeit the pad value was never defined by the user).

**Returns:**
Whether the user-defined pad value exists. It is either true or false.
- true - pad value has been specified.
- false - pad value is not specified.

Note: The system default pad value is returned if getPadValue() is called.

**Throws:**
`CDFException` - if a problem occurs checking the existence of the pad value

---

**getPadValue**

```java
public java.lang.Object getPadValue()
```

Gets the pad value set for this variable.

**Returns:**
the pad value set for this variable

---

**setSparseRecords**

```java
public void setSparseRecords(long sparseRecords)
```

Sets the sparse record type for this variable.

**Parameters:**
sparseRecords - sparse record type that should be one of the following types:
- NO_SPARSERECORDS - The variable doesn't have sparse records.
- PAD_SPARSERECORDS - The variable has pad-missing records.
- PREV_SPARSERECORDS - The variable has previous-missing records.

**Throws:**
`CDFException` - if a problem occurs setting the sparse record type

---

**getSparseRecords**

```java
public long getSparseRecords()
```
Gets the sparse record type for this variable.

**Returns:**

One of the following sparse record type is returned:

- **NO_SPARSERECORDS** - means that no sparse records are defined
- **PAD_SPARSERECORDS** - means that the variable's pad value is used when reading values from a missing record
- **PREV_SPARSERECORDS** - means that values from the previous existing records are used when reading values from a missing record

---

**setBlockingFactor**

```java
public void setBlockingFactor(long blockingFactor)
throws CDFException
```

Sets the blocking factor for this variable. The blocking factor has no effect for Non-Record varying (NRV) variables or multi-file CDFs.

**Parameters:**

- `blockingFactor` - the blocking factor - a value of zero (0) indicates that the default blocking factor should be used

**Throws:**

- `CDFException` - if a problem occurs setting the blocking factor

---

**getBlockingFactor**

```java
public long getBlockingFactor()
throws CDFException
```

Gets the blocking factor for this variable.

**Returns:**

- the blocking factor set this variable

**Throws:**

- `CDFException` - if a problem occurs getting the blocking factor set for this variable

---
**setInitialRecords**

```java
class Variable {
    public void setInitialRecords(long nRecords)
        throws CDFException {
        Sets the number of records to be written initially for this variable.

        Parameters:
        nRecords - the number of records to be written initially

        Throws:
        CDFException - if a problem occurs writing initial records
    }
}
```

**selectCacheSize**

```java
class Variable {
    public void selectCacheSize(long cacheSize)
        throws CDFException {
        Sets the number of 512-byte cache buffers to be used. This operation is not applicable for a single-file CDF.

        Parameters:
        cacheSize - the number of 512-byte cache buffers

        Throws:
        CDFException - if a problem occurs allocating cache buffers
    }
}
```

**confirmCacheSize**

```java
class Variable {
    public long confirmCacheSize()
        throws CDFException {
        Gets the number of 512-byte cache buffers defined for this variable.

        Returns:
        the number of 512-byte cache buffers set for this variable

        Throws:
        CDFException - if a problem occurs getting the number of cache buffers set for this variable
    }
}
```
**selectReservePercent**

```java
public void selectReservePercent(long reservePercent)
    throws CDFException;
```

Sets the reserve percentage to be used for this variable. This operation is only applicable to compressed z Variables. The Concepts chapter in the CDF User's Guide describes the reserve percentage scheme used by the CDF library.

**Parameters:**
- `reservePercent` - the reserve percentage to be used

**Throws:**
- `CDFException` - if a problem occurs setting a reserve percentage

**confirmReservePercent**

```java
public long confirmReservePercent()
    throws CDFException;
```

Gets the reserve percentage set for this variable. This operation is only applicable to compressed z Variables.

**Returns:**
- the reserve percentage set for this variable

**Throws:**
- `CDFException` - if a problem occurs getting the reserve percentage

**confirmPadValue**

```java
public long confirmPadValue()
    throws CDFException;
```

Checks the existence of an explicitly specified pad value for the current z variable. If an explicit pad value has not been specified, the informational status code NO_PADVALUE_SPECIFIED is returned. Otherwise, CDF_OK is returned.
Returns:
Existence of pad value. If no pad value is specified for this variable, NO_PADVALUE_SPECIFIED is returned. If a pad value has been specified, then CDF_OK is returned.

Throws:
CDFException - if a problem occurs checking the existence of pad value.

getAllocatedFrom

public long getAllocatedFrom(long recNum)
throws CDFException

Inquires the next allocated record at or after a given record for this variable.

Parameters:
recNum - The record number at which to begin searching for the next allocated record. If this record exists, it will be considered the next allocated record.

Returns:
the number of the next allocated record

Throws:
CDFException - if a problem occurs getting the number of the next allocated record

getAllocatedTo

public long getAllocatedTo(long firstRec)
throws CDFException

Inquires the last allocated record (before the next unallocated record) at or after a given record for this variable.

Parameters:
firstRec - the record number at which to begin searching for the last allocated record

Returns:
the number of the last allocated record

Throws:
**updateDataSpec**

```java
public void updateDataSpec(long dataType,
    long numElements)
    throws CDFException
```

Update the data specification (data type and number of elements) of the variable.

**Throws:**

CDFException

**getAttributes**

```java
public java.util.Vector getAttributes()
```

Returns the variable attributes that are associated with this variable.

The following example describes how to retrieve all the variable attributes that are associated with a particular variable.

```java
Variable v = cdf.getVariable("myVariable");
Vector attrs = v.getAttributes();
if (attrs.size() > 0) {
    for (Enumeration e=attrs.elements(); e.hasMoreElements();) {
        Attribute a = (Attribute) e.nextElement();
        // manipulate the attribute
    }
}
```

**Returns:**

Returns the variable attributes that are associated with this variable.
## Package gsfc.nssdc.cdf

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFConstants</td>
<td>This class defines the constants used by the CDF library and CDF Java APIs, and it mimics the cdf.h include file from the cdf distribution.</td>
</tr>
<tr>
<td>CDFDelegate</td>
<td>This class defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.</td>
</tr>
<tr>
<td>CDFObject</td>
<td>CDFObject provides the base interface for all CDF objects.</td>
</tr>
</tbody>
</table>

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>This class contains the methods that are associated with either global or variable attributes.</td>
</tr>
<tr>
<td>CDF</td>
<td>The CDF class is the main class used to interact with a CDF file.</td>
</tr>
<tr>
<td>CDFData</td>
<td>This class acts as the glue between the Java code and the Java Native Interface (JNI) code.</td>
</tr>
<tr>
<td>CDFNativeLibrary</td>
<td>This class implements the method that act as the gateway between the CDF Java APIs and the CDF library.</td>
</tr>
<tr>
<td>CDFTools</td>
<td>CDFTools.java Created: Tue Nov 24 16:14:50 1998</td>
</tr>
<tr>
<td>Entry</td>
<td>This class describes a CDF global or variable attribute entry.</td>
</tr>
<tr>
<td>Variable</td>
<td>The Variable class defines a CDF variable.</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFException</td>
<td>This class defines the informational, warning, and error messages that can arise from CDF operations.</td>
</tr>
</tbody>
</table>
# Package gsfc.nssdc.cdf.util

## Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CDFTT2000</strong></td>
<td>This class contains the handy utility routines (methods) called by the core CDF Java APIs.</td>
</tr>
<tr>
<td><strong>CDFUtils</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Epoch</strong></td>
<td><strong>Example:</strong> // Get the milliseconds to Aug 5, 1990 at 5:00 double ep = Epoch.compute(1990, 8, 5, 5, 0, 0, 0); //Get the year, month, day, hour, minutes, seconds, milliseconds for ep long times[] = Epoch.breakdown(ep); for (int i=0; i&lt;times.length; i++) System.out.print(times[i]+&quot; &quot;); System.out.println(); // Printout the epoch in various formats System.out.println(Epoch.encode(ep)); System.out.println(Epoch.encode1(ep)); System.out.println(Epoch.encode2(ep)); System.out.println(Epoch.encode3(ep)); System.out.println(Epoch.encode4(ep)); // Print out the date using format String format = &quot;, at :&quot;; System.out.println(Epoch.encodex(ep,format));</td>
</tr>
<tr>
<td><strong>Epoch16</strong></td>
<td><strong>Example:</strong> // Get the time, down to picoseconds, for Aug 5, 1990 at 5:0:0.0.0.0 double[] epoch16 = new double[2]; double ep = Epoch16.compute(1990, 8, 5, 5, 0, 0, 0, 0, 0, 0, epoch16); //Get the year, month, day, hour, minutes, seconds, milliseconds, // microseconds, nanoseconds and picoseconds for epoch16 long times[] = Epoch16.breakdown(epoch16); for (int i=0; i&lt;times.length; i++) System.out.print(times[i]+&quot; &quot;); System.out.println(); // Printout the epoch in various formats System.out.println(Epoch16.encode(epoch16)); System.out.println(Epoch16.encode1(epoch16)); System.out.println(Epoch16.encode2(epoch16)); System.out.println(Epoch16.encode3(epoch16)); System.out.println(Epoch16.encode4(epoch16)); // Print out the date using format String format = &quot;, at :&quot;; System.out.println(Epoch16.encodex(epoch16,format));</td>
</tr>
<tr>
<td><strong>EpochNative</strong></td>
<td>The Epoch class is a Java wrapper to the CDF epoch handling routines.</td>
</tr>
</tbody>
</table>
### Constant Field Values

#### gsfc.nssdc.*

<table>
<thead>
<tr>
<th>gsfc.nssdc.cdf.CDFConstants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final long</td>
<td></td>
</tr>
<tr>
<td>AHUFF_COMPRESSION</td>
<td>3L</td>
</tr>
<tr>
<td>ALPHAOSF1_DECODING</td>
<td>13L</td>
</tr>
<tr>
<td>ALPHAOSF1_ENCODING</td>
<td>13L</td>
</tr>
<tr>
<td>ALPHAVMSd_DECODING</td>
<td>14L</td>
</tr>
<tr>
<td>ALPHAVMSd_ENCODING</td>
<td>14L</td>
</tr>
<tr>
<td>ALPHAVMSg_DECODING</td>
<td>15L</td>
</tr>
<tr>
<td>ALPHAVMSg_ENCODING</td>
<td>15L</td>
</tr>
<tr>
<td>ALPHAVMSi_DECODING</td>
<td>16L</td>
</tr>
<tr>
<td>ALPHAVMSi_ENCODING</td>
<td>16L</td>
</tr>
<tr>
<td>ATTR_</td>
<td>85L</td>
</tr>
<tr>
<td>ATTR_EXISTENCE_</td>
<td>95L</td>
</tr>
<tr>
<td>ATTR_EXISTS</td>
<td>-2001L</td>
</tr>
<tr>
<td>ATTR_MAXgENTRY_</td>
<td>89L</td>
</tr>
<tr>
<td>ATTR_MAXrENTRY_</td>
<td>91L</td>
</tr>
<tr>
<td>ATTR_MAXzENTRY_</td>
<td>93L</td>
</tr>
<tr>
<td>ATTR_NAME_</td>
<td>87L</td>
</tr>
<tr>
<td>ATTR_NAME_TRUNC</td>
<td>-1001L</td>
</tr>
<tr>
<td>public static final long</td>
<td>Attr/Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ATTR_NUMBER_</td>
<td>88L</td>
</tr>
<tr>
<td>ATTR_NUMgENTRIES_</td>
<td>90L</td>
</tr>
<tr>
<td>ATTR_NUMrENTRIES_</td>
<td>92L</td>
</tr>
<tr>
<td>ATTR_NUMzENTRIES_</td>
<td>94L</td>
</tr>
<tr>
<td>ATTR_SCOPE_</td>
<td>86L</td>
</tr>
<tr>
<td>BACKWARD_</td>
<td>1010L</td>
</tr>
<tr>
<td>BACKWARDFILEoff</td>
<td>0L</td>
</tr>
<tr>
<td>BACKWARDFILEon</td>
<td>1L</td>
</tr>
<tr>
<td>BAD_ALLOCATE_RECS</td>
<td>-2015L</td>
</tr>
<tr>
<td>BAD_ARGUMENT</td>
<td>-2022L</td>
</tr>
<tr>
<td>BAD_ATTR_NAME</td>
<td>-2044L</td>
</tr>
<tr>
<td>BAD_ATTR_NUM</td>
<td>-2042L</td>
</tr>
<tr>
<td>BAD_BLOCKING_FACTOR</td>
<td>-2031L</td>
</tr>
<tr>
<td>BAD_CACHE_SIZE</td>
<td>-2063L</td>
</tr>
<tr>
<td>BAD_CDF_EXTENSION</td>
<td>-2016L</td>
</tr>
<tr>
<td>BAD_CDF_ID</td>
<td>-2002L</td>
</tr>
<tr>
<td>BAD_CDF_NAME</td>
<td>-2049L</td>
</tr>
<tr>
<td>BAD_CDFSTATUS</td>
<td>-2034L</td>
</tr>
<tr>
<td>BAD_CHECKSUM</td>
<td>-2225L</td>
</tr>
<tr>
<td>BAD_COMPRESSION_PARM</td>
<td>-2097L</td>
</tr>
<tr>
<td>BAD_DATA_TYPE</td>
<td>-2003L</td>
</tr>
<tr>
<td>BAD_DECODING</td>
<td>-2079L</td>
</tr>
<tr>
<td>BAD_DIM_COUNT</td>
<td>-2039L</td>
</tr>
<tr>
<td>BAD_DIM_INDEX</td>
<td>-2005L</td>
</tr>
<tr>
<td>BAD_DIM_INTERVAL</td>
<td>-2040L</td>
</tr>
<tr>
<td>BAD_DIM_SIZE</td>
<td>-2004L</td>
</tr>
<tr>
<td>BAD_ENCODING</td>
<td>-2006L</td>
</tr>
<tr>
<td>BAD_ENTRY_NUM</td>
<td>-2043L</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>-2058</td>
<td>BAD_FNC_OR_ITEM</td>
</tr>
<tr>
<td>-2014</td>
<td>BAD_FORMAT</td>
</tr>
<tr>
<td>-2030</td>
<td>BAD_INITIAL_RECS</td>
</tr>
<tr>
<td>-2007</td>
<td>BAD_MAJORITY</td>
</tr>
<tr>
<td>-2026</td>
<td>BAD_MALLOC</td>
</tr>
<tr>
<td>-2081</td>
<td>BAD_NEGtoPOSfp0_MODE</td>
</tr>
<tr>
<td>-2008</td>
<td>BAD_NUM_DIMS</td>
</tr>
<tr>
<td>-2011</td>
<td>BAD_NUM_ELEMS</td>
</tr>
<tr>
<td>-2036</td>
<td>BAD_NUM_VARS</td>
</tr>
<tr>
<td>-2073</td>
<td>BAD_READONLY_MODE</td>
</tr>
<tr>
<td>-2037</td>
<td>BAD_REC_COUNT</td>
</tr>
<tr>
<td>-2038</td>
<td>BAD_REC_INTERVAL</td>
</tr>
<tr>
<td>-2009</td>
<td>BAD_REC_NUM</td>
</tr>
<tr>
<td>-2010</td>
<td>BAD_SCOPE</td>
</tr>
<tr>
<td>-2111</td>
<td>BAD_SCRATCH_DIR</td>
</tr>
<tr>
<td>-2110</td>
<td>BAD_SPARSEARRAYS_PARM</td>
</tr>
<tr>
<td>-2045</td>
<td>BAD_VAR_NAME</td>
</tr>
<tr>
<td>-2041</td>
<td>BAD_VAR_NUM</td>
</tr>
<tr>
<td>-2072</td>
<td>BAD_zMODE</td>
</tr>
<tr>
<td>-2103</td>
<td>CANNOT_ALLOCATE_RECORDS</td>
</tr>
<tr>
<td>-2051</td>
<td>CANNOT_CHANGE</td>
</tr>
<tr>
<td>-2091</td>
<td>CANNOT_COMPRESS</td>
</tr>
<tr>
<td>-2104</td>
<td>CANNOT_COPY</td>
</tr>
<tr>
<td>-2100</td>
<td>CANNOT_SPARSEARRAYS</td>
</tr>
<tr>
<td>-2099</td>
<td>CANNOT_SPARSERECORDS</td>
</tr>
<tr>
<td>1</td>
<td>CDF_</td>
</tr>
<tr>
<td>201</td>
<td>CDF_ACCESS_</td>
</tr>
<tr>
<td>64</td>
<td>CDF_ATTR_NAME_LEN</td>
</tr>
<tr>
<td>Constant Name</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CDF_ATTR_NAME_LEN256</td>
<td>256L</td>
</tr>
<tr>
<td>CDF_BYTE</td>
<td>41L</td>
</tr>
<tr>
<td>CDF_CACHESIZE_</td>
<td>117L</td>
</tr>
<tr>
<td>CDF_CHAR</td>
<td>51L</td>
</tr>
<tr>
<td>CDF_CHECKSUM_</td>
<td>156L</td>
</tr>
<tr>
<td>CDF_CLOSE_ERROR</td>
<td>-2055L</td>
</tr>
<tr>
<td>CDF_COMPRESSION_</td>
<td>130L</td>
</tr>
<tr>
<td>CDF_COPYRIGHT_</td>
<td>7L</td>
</tr>
<tr>
<td>CDF_COPYRIGHT_LEN</td>
<td>256L</td>
</tr>
<tr>
<td>CDF_CREATE_ERROR</td>
<td>-2066L</td>
</tr>
<tr>
<td>CDF_DECODING_</td>
<td>4L</td>
</tr>
<tr>
<td>CDF_DELETE_ERROR</td>
<td>-2088L</td>
</tr>
<tr>
<td>CDF_DOUBLE</td>
<td>45L</td>
</tr>
<tr>
<td>CDF_ENCODING_</td>
<td>3L</td>
</tr>
<tr>
<td>CDF_EPOCH</td>
<td>31L</td>
</tr>
<tr>
<td>CDF_EPOCH16</td>
<td>32L</td>
</tr>
<tr>
<td>CDF_EXISTS</td>
<td>-2013L</td>
</tr>
<tr>
<td>CDF_FLOAT</td>
<td>44L</td>
</tr>
<tr>
<td>CDF_FORMAT_</td>
<td>6L</td>
</tr>
<tr>
<td>CDF_INCREMENT_</td>
<td>15L</td>
</tr>
<tr>
<td>CDF_INFO_</td>
<td>129L</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>1L</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>2L</td>
</tr>
<tr>
<td>CDF_INT4</td>
<td>4L</td>
</tr>
<tr>
<td>CDF_INT8</td>
<td>8L</td>
</tr>
<tr>
<td>CDF_INTERNAL_ERROR</td>
<td>-2035L</td>
</tr>
<tr>
<td>CDF_MAJORITY_</td>
<td>5L</td>
</tr>
<tr>
<td>CDF_MAX_DIMS</td>
<td>10L</td>
</tr>
<tr>
<td>Method</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>public static final long CDF_MAX_PARMS</td>
<td>5L</td>
</tr>
<tr>
<td>public static final long CDF_MIN_DIMS</td>
<td>0L</td>
</tr>
<tr>
<td>public static final long CDF_NAME_</td>
<td>2L</td>
</tr>
<tr>
<td>public static final long CDF_NAME_TRUNC</td>
<td>-1002L</td>
</tr>
<tr>
<td>public static final long CDF_NEGtoPOSfp0_MODE_</td>
<td>19L</td>
</tr>
<tr>
<td>public static final long CDF_NUMATTRL</td>
<td>10L</td>
</tr>
<tr>
<td>public static final long CDF_NUMgATTRL</td>
<td>11L</td>
</tr>
<tr>
<td>public static final long CDF_NUMrVARS</td>
<td>8L</td>
</tr>
<tr>
<td>public static final long CDF_NUMvATTRL</td>
<td>12L</td>
</tr>
<tr>
<td>public static final long CDF_NUMzVARS</td>
<td>9L</td>
</tr>
<tr>
<td>public static final long CDF_OK</td>
<td>0L</td>
</tr>
<tr>
<td>public static final long CDF_OPEN_ERROR</td>
<td>-2012L</td>
</tr>
<tr>
<td>public static final long CDF_PATHNAME_LEN</td>
<td>512L</td>
</tr>
<tr>
<td>public static final long CDF_REAL4</td>
<td>21L</td>
</tr>
<tr>
<td>public static final long CDF_REAL8</td>
<td>22L</td>
</tr>
<tr>
<td>public static final long CDF_RELEASE_</td>
<td>14L</td>
</tr>
<tr>
<td>public static final long CDF_SAVE_ERROR</td>
<td>-2083L</td>
</tr>
<tr>
<td>public static final long CDF_SCRATCHDIR_</td>
<td>149L</td>
</tr>
<tr>
<td>public static final long CDF_STATUSTEXT_LEN</td>
<td>80L</td>
</tr>
<tr>
<td>public static final long CDF_TIME_TT2000</td>
<td>33L</td>
</tr>
<tr>
<td>public static final long CDF_UCHAR</td>
<td>52L</td>
</tr>
<tr>
<td>public static final long CDF_UINT1</td>
<td>11L</td>
</tr>
<tr>
<td>public static final long CDF_UINT2</td>
<td>12L</td>
</tr>
<tr>
<td>public static final long CDF_UINT4</td>
<td>14L</td>
</tr>
<tr>
<td>public static final long CDF_VAR_NAME_LEN</td>
<td>64L</td>
</tr>
<tr>
<td>Public static final long</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CDF_VAR_NAME_LEN256</td>
<td>256L</td>
</tr>
<tr>
<td>CDF_VERSION</td>
<td>13L</td>
</tr>
<tr>
<td>CDF_WARN</td>
<td>-2000L</td>
</tr>
<tr>
<td>CDF_WRITE_ERROR</td>
<td>-2075L</td>
</tr>
<tr>
<td>CDF_zMODE</td>
<td>18L</td>
</tr>
<tr>
<td>CDFwithSTATS</td>
<td>200L</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td>1012L</td>
</tr>
<tr>
<td>CHECKSUM_ERROR</td>
<td>-2226L</td>
</tr>
<tr>
<td>CHECKSUM_NOT_ALLOWED</td>
<td>-2227L</td>
</tr>
<tr>
<td>CLOSE</td>
<td>1004L</td>
</tr>
<tr>
<td>COLUMN_MAJOR</td>
<td>2L</td>
</tr>
<tr>
<td>COMPRESS_CACHESIZE</td>
<td>155L</td>
</tr>
<tr>
<td>COMPRESSION_ERROR</td>
<td>-2093L</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>1006L</td>
</tr>
<tr>
<td>CORRUPTED_V2_CDF</td>
<td>-2028L</td>
</tr>
<tr>
<td>CORRUPTED_V3_CDF</td>
<td>-2223L</td>
</tr>
<tr>
<td>CREATE</td>
<td>1001L</td>
</tr>
<tr>
<td>CURgENTRY_EXISTENCE</td>
<td>126L</td>
</tr>
<tr>
<td>CURrENTRY_EXISTENCE</td>
<td>127L</td>
</tr>
<tr>
<td>CURzENTRY_EXISTENCE</td>
<td>128L</td>
</tr>
<tr>
<td>DATATYPE_MISMATCH</td>
<td>-2112L</td>
</tr>
<tr>
<td>DATATYPE_SIZE</td>
<td>125L</td>
</tr>
<tr>
<td>DECOMPRESSION_ERROR</td>
<td>-2092L</td>
</tr>
<tr>
<td>DECSTATION_DECODING</td>
<td>4L</td>
</tr>
<tr>
<td>DECSTATION_ENCODING</td>
<td>4L</td>
</tr>
<tr>
<td>DEFAULT_BYTE_PADVALUE</td>
<td>0</td>
</tr>
<tr>
<td>DEFAULT_CHAR_PADVALUE</td>
<td>32</td>
</tr>
<tr>
<td>DEFAULT_DOUBLE_PADVALUE</td>
<td>0.0</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td><code>public static final double</code></td>
<td><code>DEFAULT_EPOCH_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final float</code></td>
<td><code>DEFAULT_FLOAT_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final byte</code></td>
<td><code>DEFAULT_INT1_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final short</code></td>
<td><code>DEFAULT_INT2_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final int</code></td>
<td><code>DEFAULT_INT4_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DEFAULT_INT8_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final double</code></td>
<td><code>DEFAULT_REAL8_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DEFAULT_TT2000_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final char</code></td>
<td><code>DEFAULT_UCHAR_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final short</code></td>
<td><code>DEFAULT_UINT1_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final int</code></td>
<td><code>DEFAULT_UINT2_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DEFAULT_UINT4_PADVALUE</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DELETE_</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DID_NOT_COMPRESS</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>DUMMY_TT2000_VALUE</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EMPTY_COMPRESSED_CDF</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>END_OF_VAR</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH_STRING_LEN</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH_STRING_LEN_EXTEND</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH1_STRING_LEN</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH1_STRING_LEN_EXTEND</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH2_STRING_LEN</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH2_STRING_LEN_EXTEND</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH3_STRING_LEN</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH3_STRING_LEN_EXTEND</code></td>
</tr>
<tr>
<td><code>public static final long</code></td>
<td><code>EPOCH4_STRING_LEN</code></td>
</tr>
<tr>
<td>Public static final long</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>EPOCH4_STRING_LEN_EXTEND</td>
<td>32L</td>
</tr>
<tr>
<td>EPOCHx_FORMAT_MAX</td>
<td>68L</td>
</tr>
<tr>
<td>EPOCHx_STRING_MAX</td>
<td>50L</td>
</tr>
<tr>
<td>FORCED_PARAMETER</td>
<td>-1006L</td>
</tr>
<tr>
<td>gENTRY_</td>
<td>96L</td>
</tr>
<tr>
<td>gENTRY_DATA_</td>
<td>101L</td>
</tr>
<tr>
<td>gENTRY_DATASPEC_</td>
<td>100L</td>
</tr>
<tr>
<td>gENTRY_DATATYPE_</td>
<td>98L</td>
</tr>
<tr>
<td>gENTRY_EXISTENCE_</td>
<td>97L</td>
</tr>
<tr>
<td>gENTRY_NUMELEMS_</td>
<td>99L</td>
</tr>
<tr>
<td>GET_</td>
<td>1007L</td>
</tr>
<tr>
<td>GETCDFCHECKSUM_</td>
<td>1013L</td>
</tr>
<tr>
<td>GETCDFFILEBACKWARD_</td>
<td>1011L</td>
</tr>
<tr>
<td>GETCDFVALIDATE_</td>
<td>1015L</td>
</tr>
<tr>
<td>GETLEAPSECONDSENVVAR_</td>
<td>1016L</td>
</tr>
<tr>
<td>GLOBAL_SCOPE</td>
<td>1L</td>
</tr>
<tr>
<td>GZIP_COMPRESSION</td>
<td>5L</td>
</tr>
<tr>
<td>HOST_DECODING</td>
<td>8L</td>
</tr>
<tr>
<td>HOST_ENCODING</td>
<td>8L</td>
</tr>
<tr>
<td>HP_DECODING</td>
<td>11L</td>
</tr>
<tr>
<td>HP.Encoding</td>
<td>11L</td>
</tr>
<tr>
<td>HP_Encoding</td>
<td>11L</td>
</tr>
<tr>
<td>HUFF_COMPRESSION</td>
<td>2L</td>
</tr>
<tr>
<td>IBM_PC_OVERFLOW</td>
<td>-2023L</td>
</tr>
<tr>
<td>IBMPC_DECODING</td>
<td>6L</td>
</tr>
<tr>
<td>IBMPC_ENCODING</td>
<td>6L</td>
</tr>
<tr>
<td>IBMRS_DECODING</td>
<td>7L</td>
</tr>
<tr>
<td>IBMRS_ENCODING</td>
<td>7L</td>
</tr>
<tr>
<td>ILLEGAL_EPOCH_FIELD</td>
<td>-2224L</td>
</tr>
<tr>
<td>Public static final type</td>
<td>Symbol</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>double</td>
<td>ILLEGAL_EPOCH_VALUE</td>
</tr>
<tr>
<td>long</td>
<td>ILLEGAL_FOR_SCOPE</td>
</tr>
<tr>
<td>long</td>
<td>ILLEGAL_IN_zMODE</td>
</tr>
<tr>
<td>long</td>
<td>ILLEGAL_ON_V1_CDF</td>
</tr>
<tr>
<td>long</td>
<td>ILLEGAL_TT2000_VALUE</td>
</tr>
<tr>
<td>Constant Name</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>NO_DELETE_ACCESS</td>
<td>-2087L</td>
</tr>
<tr>
<td>NO_ENTRY_SELECTED</td>
<td>-2047L</td>
</tr>
<tr>
<td>NO_MORE_ACCESS</td>
<td>-2077L</td>
</tr>
<tr>
<td>NO_PADVALUE_SPECIFIED</td>
<td>1005L</td>
</tr>
<tr>
<td>NO_SPARSEARRAYS</td>
<td>0L</td>
</tr>
<tr>
<td>NO_SPARSE_RECORDS</td>
<td>0L</td>
</tr>
<tr>
<td>NO_STATUS_SELECTED</td>
<td>-2052L</td>
</tr>
<tr>
<td>NO_SUCH_ATTR</td>
<td>-2017L</td>
</tr>
<tr>
<td>NO_SUCH_CDF</td>
<td>-2067L</td>
</tr>
<tr>
<td>NO_SUCH_ENTRY</td>
<td>-2018L</td>
</tr>
<tr>
<td>NO_SUCH_RECORD</td>
<td>-2102L</td>
</tr>
<tr>
<td>NO_SUCH_VAR</td>
<td>-2019L</td>
</tr>
<tr>
<td>NO_VAR_SELECTED</td>
<td>-2048L</td>
</tr>
<tr>
<td>NO_VARS_IN_CDF</td>
<td>1006L</td>
</tr>
<tr>
<td>NO_WRITE_ACCESS</td>
<td>-2086L</td>
</tr>
<tr>
<td>NONE_CHECKSUM</td>
<td>0L</td>
</tr>
<tr>
<td>NOT_A_CDF</td>
<td>-2027L</td>
</tr>
<tr>
<td>NOT_A_CDF_OR_NOT_SUPPORTED</td>
<td>-2113L</td>
</tr>
<tr>
<td>NOVARY</td>
<td>0L</td>
</tr>
<tr>
<td>NULL_</td>
<td>1000L</td>
</tr>
<tr>
<td>OPEN_</td>
<td>1002L</td>
</tr>
<tr>
<td>OPTIMAL_ENCODING_TREES</td>
<td>0L</td>
</tr>
<tr>
<td>OTHER_CHECKSUM</td>
<td>2L</td>
</tr>
<tr>
<td>PAD_SPARSE_RECORDS</td>
<td>1L</td>
</tr>
<tr>
<td>PPC_DECODING</td>
<td>9L</td>
</tr>
<tr>
<td>PPC_ENCODING</td>
<td>9L</td>
</tr>
<tr>
<td>PRECEDING_RECORDS_ALLOCATED</td>
<td>1009L</td>
</tr>
<tr>
<td>PREV_SPARSE_RECORDS</td>
<td>2L</td>
</tr>
<tr>
<td>public static final long</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>PUT</td>
<td>1008L</td>
</tr>
<tr>
<td>READ_ONLY_DISTRIBUTION</td>
<td>-2054L</td>
</tr>
<tr>
<td>READ_ONLY_MODE</td>
<td>-2070L</td>
</tr>
<tr>
<td>READONLYoff</td>
<td>0L</td>
</tr>
<tr>
<td>READONLYon</td>
<td>-1L</td>
</tr>
<tr>
<td>rENTRY</td>
<td>102L</td>
</tr>
<tr>
<td>rENTRY_DATA</td>
<td>108L</td>
</tr>
<tr>
<td>rENTRY_DATASPEC</td>
<td>107L</td>
</tr>
<tr>
<td>rENTRY_DATATYPE</td>
<td>105L</td>
</tr>
<tr>
<td>rENTRY_EXISTENCE</td>
<td>104L</td>
</tr>
<tr>
<td>rENTRY_NAME</td>
<td>103L</td>
</tr>
<tr>
<td>rENTRY NUMELEMS</td>
<td>106L</td>
</tr>
<tr>
<td>RLE_COMPRESSION</td>
<td>1L</td>
</tr>
<tr>
<td>RLE_OF_ZEROS</td>
<td>0L</td>
</tr>
<tr>
<td>ROW_MAJOR</td>
<td>1L</td>
</tr>
<tr>
<td>rVAR</td>
<td>35L</td>
</tr>
<tr>
<td>rVAR_ALLOCATEBLOCK</td>
<td>140L</td>
</tr>
<tr>
<td>rVAR_ALLOCATEDFROM</td>
<td>143L</td>
</tr>
<tr>
<td>rVAR_ALLOCATEDTO</td>
<td>144L</td>
</tr>
<tr>
<td>rVAR_ALLOCATERECS</td>
<td>123L</td>
</tr>
<tr>
<td>rVAR_BLOCKINGFACTOR</td>
<td>51L</td>
</tr>
<tr>
<td>rVAR_CACHESIZE</td>
<td>120L</td>
</tr>
<tr>
<td>rVAR_COMPRESSION</td>
<td>137L</td>
</tr>
<tr>
<td>rVAR_DATA</td>
<td>42L</td>
</tr>
<tr>
<td>rVAR_DATASPEC</td>
<td>48L</td>
</tr>
<tr>
<td>rVAR_DATATYPE</td>
<td>37L</td>
</tr>
<tr>
<td>rVAR_DIMVARYS</td>
<td>40L</td>
</tr>
<tr>
<td>rVAREXISTENCE</td>
<td>54L</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>rVAR_HYPERDATA_</code></td>
<td>43L</td>
</tr>
<tr>
<td><code>rVAR_INITIALRECS_</code></td>
<td>50L</td>
</tr>
<tr>
<td><code>rVAR_MAXallocREC_</code></td>
<td>47L</td>
</tr>
<tr>
<td><code>rVAR_MAXREC_</code></td>
<td>46L</td>
</tr>
<tr>
<td><code>rVAR_NAME_</code></td>
<td>36L</td>
</tr>
<tr>
<td><code>rVAR_nINDEXENTRIES_</code></td>
<td>53L</td>
</tr>
<tr>
<td><code>rVAR_nINDEXLEVELS_</code></td>
<td>148L</td>
</tr>
<tr>
<td><code>rVAR_nINDEXRECORDS_</code></td>
<td>52L</td>
</tr>
<tr>
<td><code>rVAR_NUMallocRECS_</code></td>
<td>142L</td>
</tr>
<tr>
<td><code>rVAR_NUMBER_</code></td>
<td>41L</td>
</tr>
<tr>
<td><code>rVAR_NUMELEMS_</code></td>
<td>38L</td>
</tr>
<tr>
<td><code>rVAR_NUMRECS_</code></td>
<td>141L</td>
</tr>
<tr>
<td><code>rVAR_PADVALUE_</code></td>
<td>49L</td>
</tr>
<tr>
<td><code>rVAR_RECORDS_</code></td>
<td>152L</td>
</tr>
<tr>
<td><code>rVAR_RECVARY_</code></td>
<td>39L</td>
</tr>
<tr>
<td><code>rVAR_RESERVEPERCENT_</code></td>
<td>150L</td>
</tr>
<tr>
<td><code>rVAR_SEQDATA_</code></td>
<td>44L</td>
</tr>
<tr>
<td><code>rVAR_SEQPOS_</code></td>
<td>45L</td>
</tr>
<tr>
<td><code>rVAR_SPARSEARRAYS_</code></td>
<td>139L</td>
</tr>
<tr>
<td><code>rVAR_SPARSERECORDS_</code></td>
<td>138L</td>
</tr>
<tr>
<td><code>rVARs_CACHESIZE_</code></td>
<td>118L</td>
</tr>
<tr>
<td><code>rVARs_DIMCOUNTS_</code></td>
<td>33L</td>
</tr>
<tr>
<td><code>rVARs_DIMINDICES_</code></td>
<td>32L</td>
</tr>
<tr>
<td><code>rVARs_DIMINTERVALS_</code></td>
<td>34L</td>
</tr>
<tr>
<td><code>rVARs_DIMSIZES_</code></td>
<td>26L</td>
</tr>
<tr>
<td><code>rVARs_MAXREC_</code></td>
<td>27L</td>
</tr>
<tr>
<td><code>rVARs_NUMDIMS_</code></td>
<td>25L</td>
</tr>
<tr>
<td><code>rVARs_RECCOUNT_</code></td>
<td>30L</td>
</tr>
<tr>
<td>Public static final long</td>
<td>Name</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>public static final long</td>
<td>rVARs_RECDATA_</td>
</tr>
<tr>
<td>public static final long</td>
<td>rVARs_RECINTERVAL_</td>
</tr>
<tr>
<td>public static final long</td>
<td>rVARs_RECNUMBER_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SAVE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SCRATCH_CREATE_ERROR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SCRATCH_DELETE_ERROR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SCRATCH_READ_ERROR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SCRATCH_WRITE_ERROR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SELECT_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SGi_DECODING_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SGi_ENCODING_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SINGLE_FILE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SINGLE_FILE_FORMAT_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SOME_ALREADY_ALLOCATED_</td>
</tr>
<tr>
<td>public static final long</td>
<td>STAGE_CACHESIZE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>STATUS_TEXT_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SUN_DECODING_</td>
</tr>
<tr>
<td>public static final long</td>
<td>SUN_ENCODING_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TOO_MANY_PARMS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TOO_MANY_VARS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TT2000_0_STRING_LEN_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TT2000_1_STRING_LEN_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TT2000_2_STRING_LEN_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TT2000_3_STRING_LEN_</td>
</tr>
<tr>
<td>public static final long</td>
<td>TT2000_TIME_ERROR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>UNKNOWN_COMPRESSION_</td>
</tr>
<tr>
<td>public static final long</td>
<td>UNKNOWN_SPARSENESS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>UNSUPPORTED_OPERATION_</td>
</tr>
<tr>
<td>Identifier</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>public static final long VALIDATE_</td>
<td>1014L</td>
</tr>
<tr>
<td>public static final long VALIDATEFILEoff</td>
<td>0L</td>
</tr>
<tr>
<td>public static final long VALIDATEFILEon</td>
<td>-1L</td>
</tr>
<tr>
<td>public static final long VAR_ALREADY_CLOSED</td>
<td>1003L</td>
</tr>
<tr>
<td>public static final long VAR_CLOSE_ERROR</td>
<td>-2056L</td>
</tr>
<tr>
<td>public static final long VAR_CREATE_ERROR</td>
<td>-2068L</td>
</tr>
<tr>
<td>public static final long VAR_DELETE_ERROR</td>
<td>-2089L</td>
</tr>
<tr>
<td>public static final long VAR_EXISTS</td>
<td>-2025L</td>
</tr>
<tr>
<td>public static final long VAR_NAME_TRUNC</td>
<td>-1003L</td>
</tr>
<tr>
<td>public static final long VAR_OPEN_ERROR</td>
<td>-2029L</td>
</tr>
<tr>
<td>public static final long VAR_READ_ERROR</td>
<td>-2020L</td>
</tr>
<tr>
<td>public static final long VAR_SAVE_ERROR</td>
<td>-2084L</td>
</tr>
<tr>
<td>public static final long VAR_WRITE_ERROR</td>
<td>-2021L</td>
</tr>
<tr>
<td>public static final long VARIABLE_SCOPE</td>
<td>2L</td>
</tr>
<tr>
<td>public static final long VARY</td>
<td>-1L</td>
</tr>
<tr>
<td>public static final long VAX_DECODING</td>
<td>3L</td>
</tr>
<tr>
<td>public static final long VAX_ENCODING</td>
<td>3L</td>
</tr>
<tr>
<td>public static final long VIRTUAL_RECORD_DATA</td>
<td>1001L</td>
</tr>
<tr>
<td>public static final long zENTRY_</td>
<td>109L</td>
</tr>
<tr>
<td>public static final long zENTRY_DATA_</td>
<td>115L</td>
</tr>
<tr>
<td>public static final long zENTRY_DATASPEC_</td>
<td>114L</td>
</tr>
<tr>
<td>public static final long zENTRY_DATATYPE_</td>
<td>112L</td>
</tr>
<tr>
<td>public static final long zENTRY_EXISTENCE_</td>
<td>111L</td>
</tr>
<tr>
<td>public static final long zENTRY_NAME_</td>
<td>110L</td>
</tr>
<tr>
<td>public static final long zENTRY_NUMELEMS_</td>
<td>113L</td>
</tr>
<tr>
<td>public static final long zMODEoff</td>
<td>0L</td>
</tr>
<tr>
<td>public static final long zMODEon1</td>
<td>1L</td>
</tr>
<tr>
<td>public static final long zMODEon2</td>
<td>2L</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_ALLOCATEBLOCK_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_ALLOCATEDFROM_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_ALLOCATEDTO_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_ALLOCATERECS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_BLOCKINGFACTOR_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_CACHESIZE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_COMPRESSION_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DATA_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DATASPEC_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DATATYPE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DIMCOUNTS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DIMINDICES_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DIMINTERVALS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DIMSIZES_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_DIMVARYS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_EXISTENCE_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_HYPERDATA_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_INITIALRECS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_MAXallocREC_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_MAXREC_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_NAME_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_nINDEXENTRIES_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_nINDEXLEVELS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_nINDEXRECORDS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_NUMallocRECS_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_NUMBER_</td>
</tr>
<tr>
<td>public static final long</td>
<td>zVAR_NUMDIMS_</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>zVAR_NUMELEMS</td>
<td>Number of elements</td>
</tr>
<tr>
<td>zVAR_NUMRECS</td>
<td>Number of records</td>
</tr>
<tr>
<td>zVAR_PADVALUE</td>
<td>Pad value</td>
</tr>
<tr>
<td>zVAR_RECCOUNT</td>
<td>Record count</td>
</tr>
<tr>
<td>zVAR_RECINTERVAL</td>
<td>Record interval</td>
</tr>
<tr>
<td>zVAR_RECNUMBER</td>
<td>Record number</td>
</tr>
<tr>
<td>zVAR_RECORDS</td>
<td>Number of records</td>
</tr>
<tr>
<td>zVAR_RECVARY</td>
<td>Send vector</td>
</tr>
<tr>
<td>zVAR_RESERVEPERCENT</td>
<td>Reserve percent</td>
</tr>
<tr>
<td>zVAR_SEQDATA</td>
<td>Sequence data</td>
</tr>
<tr>
<td>zVAR_SEQPOS</td>
<td>Sequence position</td>
</tr>
<tr>
<td>zVAR_SPARSEARRAYS</td>
<td>Sparse arrays</td>
</tr>
<tr>
<td>zVAR_SPARSERECORDS</td>
<td>Sparse records</td>
</tr>
<tr>
<td>zVARs_CACHESIZE</td>
<td>Cache size</td>
</tr>
<tr>
<td>zVARs_MAXREC</td>
<td>Maximum record</td>
</tr>
<tr>
<td>zVARs_RECDATA</td>
<td>Record data</td>
</tr>
<tr>
<td>zVARs_RECNUMBER</td>
<td>Record number</td>
</tr>
</tbody>
</table>

**gsfc.nssdc.cdf.CDFTools**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_VALUES</td>
<td>All values</td>
<td>3</td>
</tr>
<tr>
<td>NAMED_VALUES</td>
<td>Named values</td>
<td>4</td>
</tr>
<tr>
<td>NO_REPORTS</td>
<td>No reports</td>
<td>0</td>
</tr>
<tr>
<td>NO_VALUES</td>
<td>No values</td>
<td>0</td>
</tr>
<tr>
<td>NRV_VALUES</td>
<td>NRV values</td>
<td>1</td>
</tr>
<tr>
<td>REPORT_ERRORS</td>
<td>Report errors</td>
<td>1</td>
</tr>
<tr>
<td>REPORT_INFORMATION</td>
<td>Report information</td>
<td>4</td>
</tr>
<tr>
<td>REPORT_WARNINGS</td>
<td>Report warnings</td>
<td>2</td>
</tr>
<tr>
<td>RV_VALUES</td>
<td>RV values</td>
<td>2</td>
</tr>
</tbody>
</table>
gsfc.nssdc.cdf

Interfaces
CDFConstants
CDFDelegate
CDFObject

Classes
Attribute
CDF
CDFData
CDFNativeLibrary
CDFTools
Entry
Variable

Exceptions
CDFException
Classes
CDFTT2000
CDFUtils
Epoch
Epoch16
EpochNative
Package gsfc.nssdc.cdf

Class gsfc.nssdc.cdf.CDFException extends java.lang.Exception implements Serializable

Serialized Fields

myStatus

long myStatus